www.freesoftwaremagazine.com

Issue 5 - June, 2005

Free Software MAGAZINE

FOCUS:
VIRTUAL SERVERS AND FREE SOFTWARE

THE LEAP FROM VIRTUAL HOST TO VIRTUAL MACHINE
(EDWARD MACNAGHTEN)

XEN, THE VIRTUAL MACHINE MONITOR
(MOSHE BAR)

LINUX-VSERVER
(HERBERT PÖTZL)

(JOHN GOERZEN)
Haskell
A VERY DIFFERENT LANGUAGE

(KAUSTUBH GHOSH)
The future of computing: is free software ready?
BE IT ON-DEMAND OR IN-HOUSE, FREE SOFTWARE WORKS FOR ALL

(SAQIB ALI)
Newsflash!
RSS: BEYOND NEWS SITES AND WEBLOGS

(DAVID M. BERRY, GILES MOSS)
The commons without commonalty
IS THE CREATIVE COMMONS MISSING SOMETHING?

(JOHN LOCKE)
Who's behind that web site?
SSL, CERTIFICATES, AND DETECTING PHISHERS

qreate
COVER BY WWW.QREATE.NET

ISSN 1746-8752

9 771746 875009 05

Are your systems *OPEN* for business?

Our customers' are!

For over 10 years, OCI has been helping organizations build their mission critical infrastructure with open architectures using standards-based OO technologies.

OCI provides the following commercial-grade products to the open-source community:

- **ACE** - The high performance C++ portability layer combining socket-level performance with type-safe programming, across a wide range of operating systems.
- **TAO** - An open-source C++ CORBA 2.6 implementation with the widest range of services and span of platforms of any ORB.
- **JacORB** - The open-source Java CORBA 2.4 implementation.
- **MPC** - An open-source cross-platform build tool - "script once, build many"
- **OVATION** - A tool for instrumenting & visualizing CORBA systems behavior.

Can't wait to go open? OCI offers a comprehensive set of migration support services that will help your organization minimize risk, cost, and time when moving to these open-source solutions.

To learn more about these products & services, visit: www.theaceorb.com

Product Development • Consulting • Educational Services

Object Computing, Inc. (OCI)

St. Louis, MO Headquarters: +1 (314) 579-0066
Phoenix, AZ Office: +1 (480) 752-0042

www.ociweb.com

OBJECT COMPUTING, INC.

Freelock Computing

The Open Source for Business Solutions

Open Source is about solving problems. We solve business problems for small and growing businesses in the Puget Sound area. We provide technology strategy consulting, implementation, and hosting. We work with proven open source software. Go to freelock.com for details.

- Accounting Systems
- Sales Management Systems (including SugarCRM!)
- Project Management Systems
- Customer Management Systems
- Document Management Systems
- Custom web applications
- Network management
- Linux, Mac, and Windows support
- Service Co ntracts

We Wrote The Book!

"'Open Source Solutions for Small Business Problems' is a very highly recommended book for anyone who is looking at the open source market ... This is easily one of the best Linux books of the year; providing a management level view of the Linux world without the technical focus of other books."

– Harold McFarland, Readers Preference Reviews

Contact us for a free technology assessment and project roadmap for your business!

Contact us at:
info@freelock.com
1-206-579-4836

Freelock LLC
www.freelock.com

Free Software MAGAZINE

Contents

Issue 5, June 2005

EDITORIAL

The internet's plague: spam 7
Is it really going to stay with us forever?

STARTERS

Linux in a Windows World *by Roderick Smith* 8
by Martin C Brown

From Bash to Z Shell *by Oliver Kiddle* et al. 10
by Martin C Brown

FOCUS

The leap from virtual host to virtual machine 13
by Edward Macnaghten
Virtualization and implications of free software

Xen, the virtual machine monitor 19
by Moshe Bar
The art of virtualization

Linux-VServer 25
by Herbert Pötzl, Micah Anderson, Björn Steinbrink
Resource efficient context isolation

TECH WORLD

Haskell 28
by John Goerzen
A very different language

Newsflash! 34
by Saqib Ali
RSS: beyond news sites and weblogs

Web site blocking techniques 37
by Tedi Heriyanto
How to use Squid and squidGuard to restrict user access to undesirable web sites

Who's behind that web site? 45

by John Locke

SSL, certificates, and detecting phishers

WORD WORLD

The future of computing: is free software ready? 52

by Kaustubh Ghosh

Be it on-demand or in-house, free software works for all

Free software 2.0 55

by Nathaniel Palmer

Commercial class software with free license flexibility

On the "Creative Commons": a critique of the commons without commonalty 63

by David M. Berry, Giles Moss

Is the Creative Commons missing something?

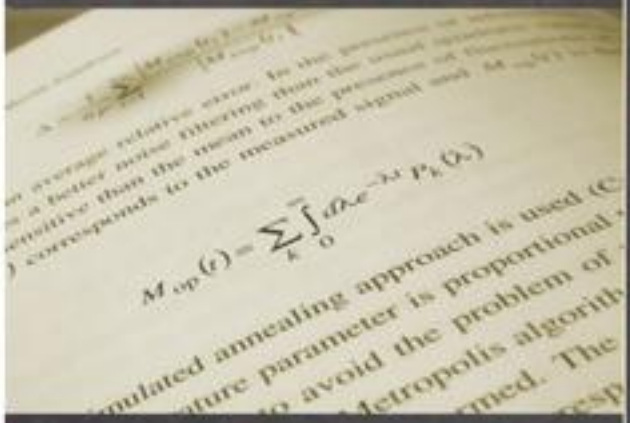
Free Software MAGAZINE

BY SUBSCRIBING YOU WILL BE SUPPORTING A MAGAZINE WHICH BELIEVES IN FREE SOFTWARE. ALL OUR ARTICLES ARE RELEASED UNDER THE GNU FREE DOCUMENTATION LICENSE, ENHANCING EXISTING INFORMATION ON FREE SOFTWARE.

SUBSCRIBE NOW!

WWW.FREESOFTWAREMAGAZINE.COM/SUBSCRIBE

Communicating mathematics...



...is getting easier.

Math Focus

News
Discussion
Collaboration
Software
Books

www.mathforge.net

Powerful tools for online mathematics

mathforge

The internet's plague: spam

Is it really going to stay with us forever?

When the internet became a "thing" for the masses, it was around 1995. Well, it was a little earlier for some, and later for some others, but I think 1995 is a pretty good point of reference.

At the time, we all thought the internet could be a *utopia*, a place where nothing really bad could happen because we were all connected to one another - almost literally.

Anonymity made things even more exciting: there was the freedom to be however we wanted to be (who has never, ever lied on IRC?!?) and to join groups we'd never dreamed of joining before.

I remember being crazy about "Queen", but none of my friends were; well, thanks to the internet I found myself (virtually) surrounded by all of these people who, like me, loved Queen! Yes, I had discovered *alt.music.queen*... I know, it doesn't sound all that ground-breaking now, but at the time doing something like that was really amazing.

But then, something changed. Spam came along and it became part of the equation.

I have always been a good dreamer, but never a good forecaster.

I discovered later (we all did) that spam was to be the internet's worst disease. Let me give a couple of examples.

Newsgroups were fantastic, because they were both centralised (there was only one *alt.music.queen*) and distributed (there were several servers which could feed the same group). A new newsgroup was created if enough people wanted it. Thanks to this fantastic system, online communities were created easily and people could easily find others with the same interests. Unfortunately, because of spam, newsgroups have died (and managed to resurrect) a couple of times over the last two years. Online forums and mailing lists today play what used to be the newsgroups' role, but they struggle to attract people the way newsgroups did because of the lack of "centralisation".

IRC was fantastic too, especially if you didn't have a job. When IRC started having overcrowding problems, more IRC networks were created. It looked like things could actually work out. Overcrowding didn't seem to be too much of an issue anymore. Then, spam came along - yes, IRC spam! For many people (including me), that was the end of it. I haven't spent time on IRC for years, because every time I tried I was put off by spam messages, even in the most "serious" network.

And there is email. I believe the reason that email still exists is because it's become so insanely necessary in today's world. Without email, there would be no Free Software Magazine. Email is doing everything possible – everything – to survive. It's a hard battle to survive, and I do wonder sometimes if it will manage to avoid being destroyed - by idiotic patent disputes by Microsoft, or by changes in the protocol, which make it harder and more expensive to use.

What about the World Wide Web? Is it really spam free? Well, it is as long as you don't let your users collaborate to your page's contents. If you have a wiki, or a guestbook, or whatever, then you will see: yes, even the www is spammed.

The depressing thing is that nobody seems to be able to come up with a "solution" to the spam problem. I sat down and thought about it for hours and hours. Many people must have done it. None of them have found a way out. (While these people were thinking, Microsoft continued its unbearable behaviour and tried to play the usual "it's my patent" game and fortunately it hasn't managed to win yet).

At this point, it's likely that spam is going to be like the cold virus – something humanity simply has to put up with, without ever "fixing" it. If that's the case, our role (as free software advocates) is to make sure that no single company (especially monopolists) has exclusive rights to use widely adopted technologies to fight spam.

Let the fight continue...

Copyright information © 2005 by Tony Mobily
Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

Free Software Magazine is a magazine by **The Open Company Partners Inc.**, 90 Main St. Road Town, Tortola BVI

EDITOR IN CHIEF
Tony Mobily (t.mobily@)

TECHNICAL EDITORS
Clare James (c.james@)
Pancrazio De Mauro (p.demauro@)

EDITORS
Anna Dymitr Hawkes (a.dymitrhawkes@)
Dave Guard (d.guard@)

ADVERTISING AND SALES
Stephen Wetzel (s.wetzel@)

TECHS
Gianluca Pignalberi (LaTeX class and magazine generation) (g.pignalberi@)
Gian Maria Ricci (RTF to XML converter using VBA) (gm.ricci@)

GRAPHIC DESIGN
Alan Sprecacenere (Web, cover and advertising design) (a.sprecacenere@)
Tony Mobily, Gianluca Pignalberi, Alan Sprecacenere (Magazine design)

CONTRIBUTORS
Saqib Ali, Micah Anderson, Moshe Bar, David M. Berry, Martin C Brown, Kaustubh Ghosh, John Goerzen, Tedi Heriyanto, John Locke, Edward Macnaghten, Giles Moss, Nathaniel Palmer, Björn Steinbrink, Herbert Pötzl,

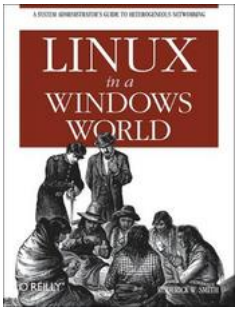
THIS PROJECT EXISTS THANKS TO
Donald E. Knuth, Leslie Lamport, People at TeX Users Group TUG (http://www.tug.org)

Every listed person is contactable by email. Please just add freesoftwaremagazine.com to the person's username in parentheses.

For copyright information about the contents of Free Software Magazine, please see the section "Copyright information" at the end of each article. Unless the license is applied immediately with the writing "(The following license is effective immediately)", for six weeks after publication you may not reproduce or retransmit the article, in whole or in part, in any manner, without the prior written consent of the author; six weeks after publication, the license at the end of each article applies.

Linux in a Windows World *by Roderick Smith*

Martin C Brown



Linux in Windows World aims to solve the problems experienced by many system administrators when it comes to using Linux servers (and to a lesser extent clients) within an existing Windows environment. Overall the book is meaty and a quick flick through shows an amazing amount of information has been crammed between the covers. There are though some immediately obvious omissions, given the books title and description, but I'm hoping this won't detract from the rest of the content.

The contents

The book starts off with a look at where Linux fits into a Windows network, covering its use both as a server and desktop platform. Roderick makes some salient points and arguments here, primarily for, rather than against, Linux but he's not afraid to point out the limitations either. This first section leads on to a more in depth discussion of deploying a Linux system into your network, promoting Linux in a series of target areas – email serving, databases and so on – as well as some strategies for migrating existing Windows desktops to Linux.

It's great to see the often forgotten issue of backups getting a chapter of its own and the extensive information on authentication solutions are invaluable

The third chapter and the start of the second section starts to look in detail at the various systems and hurdles faced through using Linux within an existing heavily Windows focused environment. This entire section is primarily devoted to Samba and sharing and using shared files and printers.

Section 3 concentrates on centralized authentication, including using LDAP and Kerberos in place of the started Windows and Linux solutions.

Remote login, including information on SSH, Telnet and VNC make up content of the fourth section. Most useful among the chapters is the one on Remote X Access which provides vital information on X server options for Windows, and information on configuring XDMCP for session management.

The final section covers the installation and configuration of Linux based servers for well-known technologies such as email, backups and network manage (DNS, DHCP etc).

Who's this book for?

Overall, the tone of the book is geared almost entirely towards administrators deploying Linux as a server solution and migrating your Windows clients to using the Linux server. The "integration" focus of the book concentrates on replacing Windows servers with Linux equivalents, rather than integrating Linux servers and clients into an existing Windows installation.

All these gaps make the book a "Converting your Windows World to Run on Linux Servers" title, rather than what the book's title (and cover description) suggests. If you are looking for a book that shows you how to integrate your Linux machines into your Windows network, this book won't help as much as you might have hoped.

On the other hand, if you are a system administrator and you are looking for a Windows to Linux server migration title then this book will prove invaluable. There are gaps, and the book requires you to have a reasonable amount of Linux knowledge before you start, but the information provided is excellent and will certainly solve the problems faced by many people moving from the Windows to a Linux platform.

Pros

There's good coverage here of a wide range of topics. The information on installing and configuring Linux equivalents of popular Windows technologies is very nice to see, although I would have preferred some more comparative information between the way Windows and the Linux counterparts work and operate these solutions.

Some surprising chapters and topics also shine through. It's great to see the often forgotten issue of backups getting a chapter of its own and the extensive information on authentication solutions are invaluable.

Cons

I found the organization slightly confusing. For example, Chapter 3 is about using Samba, but only to configure Linux as a server for sharing files. Chapter 4 then covers sharing your Linux printers to Windows clients. Chapter 6 then covers the use of Linux as a client to Windows for both printer and file shares. Similarly, there is a chapter devoted to Linux Thin Client configurations, but the use of *rdesktop*, which interfaces to the Windows Terminal Services system, has been tacked on to the end of a chapter on using VNC.

There are also numerous examples of missed opportunities and also occasionally misleading information. Windows Server 2003 for example has a built in Telnet server and incorporates an extensive command line environment and suite of administration tools, but the book fails to acknowledge this. There's also very little information on integrating application level software, or the client-specific integration between a Linux desktop and Windows server environment. A good example here is the configuration of Linux mail clients to work with an existing Exchange Server, which is quite happy to work with standard IMAP clients. Instead, the book suggests you replace Exchange with a Linux-based alternative, and even includes solutions for configuring this solution.

Finally, there are quite a few obvious errors and typos – many of which are in the diagrams that accompany the text.

In short

Title	Linux in a Windows World
Author	Roderick W Smith
Publisher	O'Reilly
ISBN	0596007582
Year	2005
Pages	478
CD included	No
Mark (out of 10)	8

Copyright information

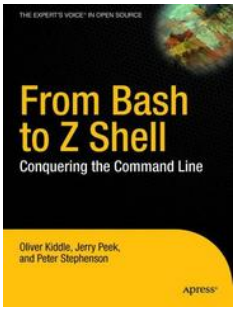
© 2005 by

This article is made available under the "Attribution-NonCommercial-NoDerivs" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-nc-nd/2.0/.

A PLANT NEEDS WATER TO GROW
Free Software
MAGAZINE
BY SUBSCRIBING YOU WILL BE SUPPORTING A MAGAZINE WHICH BELIEVES IN FREE SOFTWARE. ALL OUR ARTICLES ARE RELEASED UNDER THE GNU FREE DOCUMENTATION LICENSE, ENHANCING EXISTING INFORMATION ON FREE SOFTWARE.
SUBSCRIBE NOW!
WWW.FREESOFTWAREMAGAZINE.COM/SUBSCRIBE
FREE SOFTWARE MAGAZINE
NEEDS SUBSCRIBERS TO PROSPER!

From Bash to Z Shell *by Oliver Kiddle et al.*

Martin C Brown



If you use a free software operating system or environment, chances are one of your key interfaces will be through some kind of shell. Most people assume the bulk of the power of shells comes from the commands available within them, but some shells are actually powerful in their own right. Many of the more recent releases being more like a command line programming environment than a command line interface. "From Bash to Z Shell" published by Apress (http://www.apress.com), provides a guide to using various aspects of the shell. From the basic command line interaction through to the more complex processes of programming, it touches on file pattern matching and command line completion along the way.

The contents

Shells are complicated – how do you start describing working with a shell without first describing how the shell works, and don't you show them how to use it by doing so? The book neatly covers this problem in the first chapter with what must be the best description of a shell and how the interaction works that I've ever read.

This first chapter leads nicely into the first of three main sections. The initial section looks at using a shell, how to interact with the programs which are executed by the shell and how to use shell features such as redirection, pipes and command line editing. Other chapters look at job and process control, the shell interface to directories and files, as well as prompts and shell history.

> After you've customized your environment, extended your completion routines and enhanced your command-line once, you'll forever find yourself tweaking and optimizing the environment even further

The real meat of the book for me lies in the two main chapters in the middle that make up the second section. The first of these chapters is on pattern matching. Everybody knows about the basics of the asterisk and question mark, but both bash and zsh provide more complex pattern matching techniques that enable you to very find a specific set of files which can simplify your life immensely. The second chapter is on file completion; press TAB and get a list of files that matches what you've started to type. With a little customization you can extend this functionality to also include variables, other machines on your network and a myriad of other potentials. With a little more work in zsh and you can adjust the format and layout of the completion lists and customize the lists according to the environment and circumstances.

The third and final section covers the final progression of shell use from basic interaction to programming and extending the shell through scripts. Individual chapters cover the topics of variables, scripts and functions. The penultimate chapter puts this to good use by showing you how to write editor commands – extensions to zsh that enhance the functionality of the command line editor. Full examples and descriptions are given here on a range of topics, including my favourite: spelling correction.

The final chapter covers another extension for the command-line – completion functions. Both bash and zsh provide an extension system for completion. Although the process is understandably complex, the results can be impressive.

Who's this book for?

If you use a shell – and let's face it, who doesn't – then the information provided in the book is invaluable. Everybody from system administrators through developers and even plain old end users are going to find something in this book that will be useful to them.

Of all the target groups, I think the administrators will get the most benefit. Most administration involves heavy use of the shell for running, configuring and organizing your machine, and the tricks and techniques in this book will go a long way to simplify many of the tasks and processes that take up the time. Any book that can show you how to shorten a long command line from requiring 30-40 key presses down to less than 10 is bound to be popular.

Pros

The best aspect of the book is that it provides full examples, descriptions and reasoning for the different techniques and tricks

portrayed. This translates the content from more than a simple guide and into an essential part of the users desktop guides. The book is definitely not just an alternative way of using the online man pages.

The only problem – although it's a good one – is that reading the book and following the tips and advice given becomes addictive. After you've customized your environment, extended your completion routines and enhanced your command-line once, you'll forever find yourself tweaking and optimizing the environment even further.

Finally, it's nice to see a handy reference guide in one of the appendices to further reading – much of it online, but all of it useful.

Cons

One of the odd things about the book is that the title doesn't really reflect the contents. If you are expecting the book to be guide to using a range of shells 'From Bash to Z Shell', as the name suggests, you'll be disappointed. Sure, a lot of the material is generic and will apply to many of the shells in use today, but the bulk of the book focuses on just the two shells described in the title, which makes the title a little misleading.

Although I'm no fan of CDs in books, I would have liked to see a CD or web link to some downloadable samples from the book.

In short

Title	From Bash to Z Shell
Author	Oliver Kiddle, Jerry Peek and Peter Stephenson
Publisher	Apress
ISBN	1590593766
Year	2005
Pages	472
CD included	No
Mark (out of 10)	9

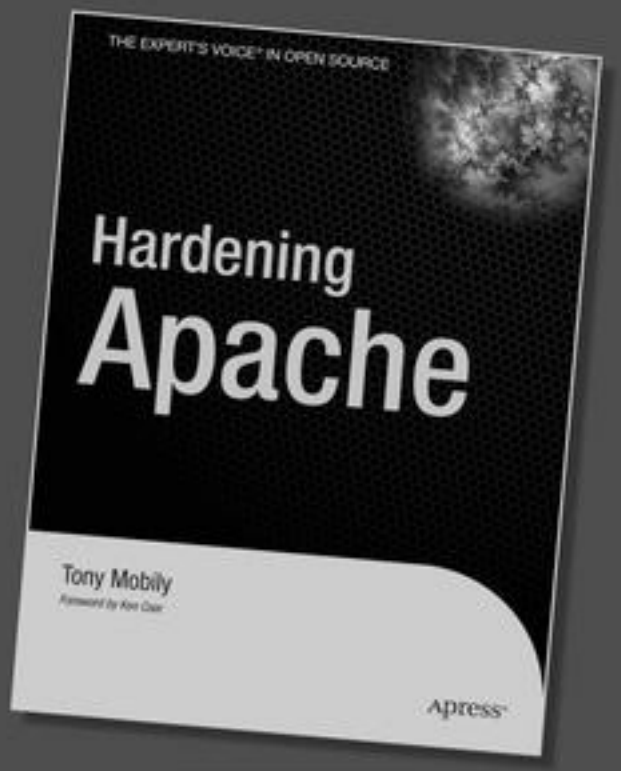
Copyright information

© 2005 by Martin C Brown

This article is made available under the "Attribution-NonCommercial-NoDerivs" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-nc-nd/2.0/.

PROTECT YOURSELF AND YOUR SERVER

Get *Hardening Apache*



Published April 2004

ISBN: 1-59059-378-2• $29.99 US • 296 pages

"This is a book that should definitely be included on any serious Apache administrator's bookshelf."

—Gianluca Insolvibile, Slashdot contributor

Throughout *Hardening Apache*, renowned author Tony Mobily introduces many of the security problems you will inevitably stumble across, and explains techniques to secure and harden your Apache server.

For more information about *Hardening Apache* or other Apress titles, please visit **www.apress.com**. Apress books are available at fine bookstores worldwide.

Apress®

qreate

Graphic, Web, Life design

PRINT

Free Software
MAGAZINE
CREATING FREE SOFTWARE MAGAZINE
A COMPLETE AUTOMATED SYSTEM TO CREATE MAGAZINES BASED ON FREE SOFTWARE
INTERVIEW WITH MYTHIC BEASTS' STAFF
THE SECOND MAC REVOLUTION
THE CONTENT TAIL WAGS THE IT DOG

FREE SOFTWARE MAGAZINE'S ISSUE 0 COVER

FREE SOFTWARE MAGAZINE'S MEDIA KIT

FREE SOFTWARE MAGAZINE'S ADVERTISEMENT

WEB

COLLE MORO BED&BREAKFAST
www.collemoro.com

VILLA DELLE RONDINI
www.villadellerondini.it

www.qreate.net ✵ *info@qreate.net*

The leap from virtual host to virtual machine

Virtualization and implications of free software

Edward Macnaghten

Back in the good old days, when men were men, women were women and the standard way for two computers to talk to each other was through a cable plugged into the serial port, was when I first took the plunge into this "internet" thingy and signed up with an ISP. Then, armed with a modem, a telephone line that doubled as a fax, Netscape 1.1 and a sense of adventure, I surfed web sites, emailed the few others I knew who had also taken the plunge and joined in on worldwide discussions on what we called "News Groups". I felt at the time I was at the sharp end of technology, that I had arrived at the next millennium early (this was still the nineties) and I had boldly gone into this brave new world of global communications where few had gone before. In a way, I had, when you consider what others around me in the UK were doing at that time.

First virtual experience

Soon after I had joined, my ISP decided to give all of its customers the ability to publish web pages for personal or business use. To announce it they came up with this excellent slogan – "Don't just surf, make waves!". They didn't give us all our own server, of course, but some space on one of their servers they had implemented for the purpose. A staggering 512 kilobytes! An entire half meg! Not only that but they gave us use of a few CGI scripts to enable functionality like the emailing of results from forms that I could write and publish on my website. Wow!

This was, I suppose, my first experience of virtualization, although I didn't think of it as such at the time. We (at first) did not have our own domain names, none the less the concept was born: that a large number of people could use an area of a single machine totally independent of others and could treat that area as their own, especially as far as anyone visiting is concerned. In short, the distinction between network and server had been blurred; someone could visit my site and, unless they looked at the URL at the top of the browser, would be unaware that I was sharing the hardware with possibly hundreds of others.

> Virtual hosts have low administration overhead, Virtual Servers have more virtual functionality, both enjoy minimal performance hit, neither offers complete virtual control

Technical advancements increased and soon the company I helped to run obtained a virtual host, similar to, as we think, one of today. It consisted of our own domain name, shell access to our area of the machine and giving me the ability to create my own web scripts, and perform email forwarding, mailing list management. It also came with all of the other paraphernalia we have come to expect from virtual host suppliers. This suited us perfectly because, for a relatively small financial outlay, we could produce a web site that, as far as most of the world's population was concerned,

The cost of non-virtualization installations could be prohibitive

was on par with the multi-million dollar sites of the corporations.

Virtual comparisons and progress

As the company grew so did our web requirements, and soon we needed a dedicated machine. We couldn't justify the cost of getting a fast link to our offices – which were temporary in any case – so we co-located it with a host provider who specialised in that. We finally had "root" administrator access on our own machine and could install whatever software we considered necessary to run our business, without the limitations a virtual host restricted us to. Freedom to do what we wanted to at last! This, however, cost us considerably more money than the shared hosting that we had before.

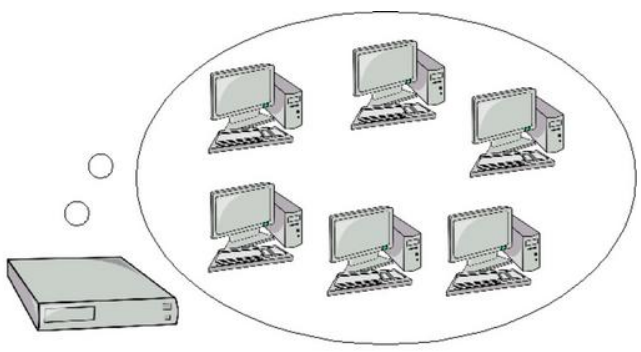
Recently a significant technical advance has occurred with web servers on the market: onto the scene has come virtual machines. I currently use one of these as my personal web site. In effect a hosting company can give you the functionality of a co-located machine for the price of a virtual host. I have root administrator access to my machine and the facility to reboot it whenever I require. I can even connect to a special server using SSH and have access to the console of my machine, should I mess up the firewall configuration, which I found useful when I put the default firewall policy as "deny" and flushed the rules (oops!). I can install any software I like and even set up virtual hosts on it. I could even, if I wanted, choose which Linux® distribution and kernel to use. All of this even though I share this machine with about fifty other people, all of whom are enjoying the same functionality for their own virtual machine, and none of us has access to each other's machines at all. The distinction between "network" and "machine" is becoming more and more obfuscated. The implications to the small web server writer are enormous. They now have the facility to install whatever databases they like, and provide whatever services they like for the monthly price of a dinner at a mediocre restaurant. An equaliser between the corporations and the one-man bands indeed.

Free software has had an important role to play in this. In fact, more than just important. If the free software model didn't exist, I can say with near certainty that virtual machines would not exist in the way they do today. Most virtual machines currently only exist in the free software world, on Linux especially. I believe that this is one area where Microsoft® cannot claim that they out-perform Linux no matter what "independent" reports they commission for the simple reason that they have yet to produce a viable virtual web-server machine.

If the free software model didn't exist, I can say with near certainty that virtual machines would not exist in the way we know them today

The way virtualization has advanced is very much on par with the ethos of free software. It is, after all, a tweak and

A large number of web pages exist on some kind of virtual machine, server or host



Different types of virtual models exist for different needs

an enhancement of the kernel and operating system. In order to successfully create a virtual machine program you need access to the magical inner workings of the machine and all of associated documentation. For proprietary systems this is often the exclusive knowledge, and always under the control of, the software vendor, and thus there exists an immediate obstacle to the development of such products – and "Why?". The answer to "Why Not" is obvious. It creates support issues. It could well lead to the sale of fewer licenses, not more, it can create performance issues, and so on. Also, in the proprietary world, one of the major benefits of virtualization disappears. It's difficult to see how a vendor would permit two instances of the operating system to be sold for any less on a virtual machine than on a real one. With this and the extra licensing they'd charge for the virtualizing software itself, a web hoster may well have to pay as much, or more, to provide a virtual machine as it would to provide a real one.

The free software version doesn't have these handicaps. This means that when someone needed or simply wanted to write virtualizing software, all of the tools and information were at hand. This means that projects like User-Mode Linux (http://user-mode-linux.sourceforge.net) could get off the ground, and be implemented and enjoyed without anyone's licensing terms being violated, and the only people being harmed are the proprietary operating systems in the market, whose vendors have seen their share of it being reduced slightly more.

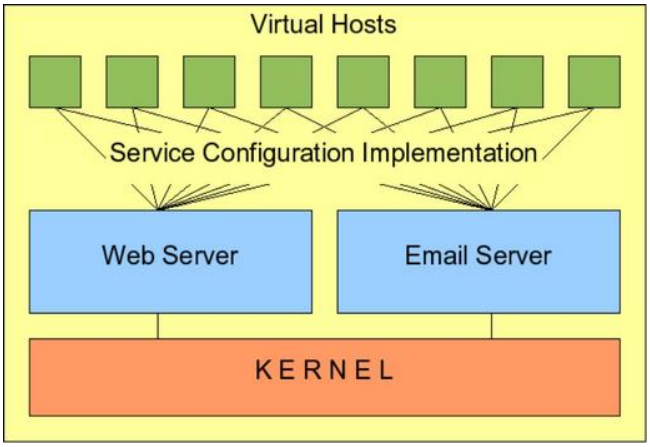
Virtualization types and advancements

As often is the case, the free software model doesn't stop there. Virtualization is about to go to the next level. In the world of virtualization, a new kid has arrived on the block that is getting noticed. And the name of the kid? Xen. To understand why it is so significant we need to examine the different types of virtualizing programs. The names I have given to these categories are my own.

1. Virtual hosts

This is not really a virtual system, just a refinement of service configurations. The web and email services are configured so that more than one domain can be hosted, though each of the virtual areas of the machine are independent of each other, they all share the same kernel. Web modules, database engines and the programs installed are decided by the administrator of the main machine (also known as the "monitor" in virtualization parlance). Also, it's often the case that all the virtual machines share the same IP address. This is the type of virtualization that most web hosting companies currently supply. The advantage of this is that it is relatively cheap and easy to set up – you can even do this on Microsoft's Servers – also there's no resource or performance hit from the virtualization itself, mainly because there isn't really any. The disadvantage of this is that, although you have the cost savings of virtualization, the virtual machine owners do not have the benefits of their own machine. You cannot control what software goes on it. You do not have administrator or root access at all, so you can't even change the time. Often, for these machines, you don't even have command-line shell access. You are, to all intents and purposes, totally at the mercy of the web hosting company.

In the "virtual host" model all members share the same services



2. Virtual servers

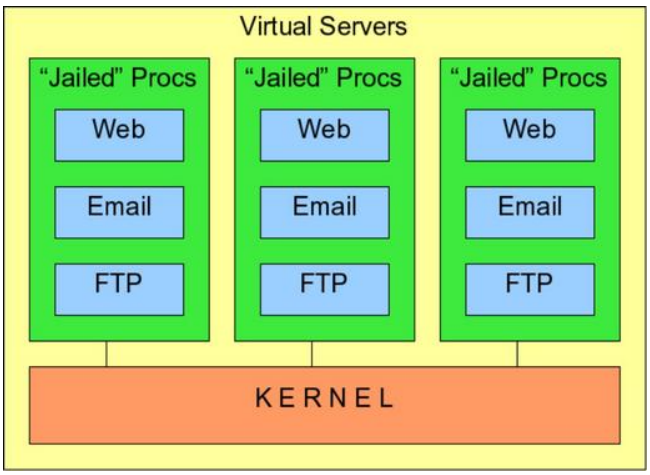
Although the mechanics of this are similar to the "Virtual Host", described above, it offers a far greater level of virtualization. The Linux Vserver (http://linux-vserver.org/) project is an example of this. The basis of this is the ability POSIX operating systems, such as Linux, have for a process to be "chrooted", or to run in "jails". That is, a process can be started in a particular directory, in a manner that it thinks that the directory is "root", or the top of the entire file system. It cannot even see the rest of the file system, let alone access it. An analogy of this is for a boss of a large company giving an employee access to only one floor of an office block and telling him that the building only has one floor – that one. And him believing it! Each virtual server has a directory – or a floor if you like – and sub directories (offices) containing the necessary programs and configuration files required for it. For each "floor" a master process is created and "chroot"ed into its very own directory, and all services for that virtual server – such as web and email – are "forked" or started by that master process, thus inheriting the same restrictions. This means that each virtual machine can run its own web, email, ftp and database server independently. However, they share the same kernel, so as with the "Virtual Host" model, their is no performance hit in the virtualization itself. Each virtual server owner does have the benefit of much more control over their own "server" by comparison with the "virtual hosts" above, in so far as they can install and configure their own software. Although the control is often enough, it is not extensive. As a "virtual server" owner, it is difficult to configure your own firewall, and there is no way to mess about with the kernel or its parameters either. And you still can't change the time.

3. User mode virtual machines

An example of this is User Mode Linux (http://user-mode-linux.sourceforge.net). Here, certain resources, such as the amount of memory, disk space and so on, are assigned to each machine in a configuration file. The main machine, or "monitor", fires off a process for each virtual machine. This process is a wrapper that runs a kernel resident on the virtual machine itself. This means that the owner of each virtual machine has, more or less, complete control of that machine, down to which kernel is used.

Functionally, it is almost identical to having your own co-located real machine, with the added benefit of the web hoster being able to provide a special URL where you can SSH to perform administration tasks such as rebooting and grant you access to your virtual system consoles. Also, finally you can change the time on your virtual machine! All of this comes at a price however. Your virtual machine's kernel is in fact running as a user process on the master machine. This means there is a severe performance hit in the virtualization itself. Although this is often easily counteracted with fast hardware being relatively cheap, it none the less is an unwelcome attribute. On a side note Cooperative Linux (http://www.colinux.org) also falls into this category, this is where you can run a number of Linux virtual machines as processes on a master Microsoft Windows machine.

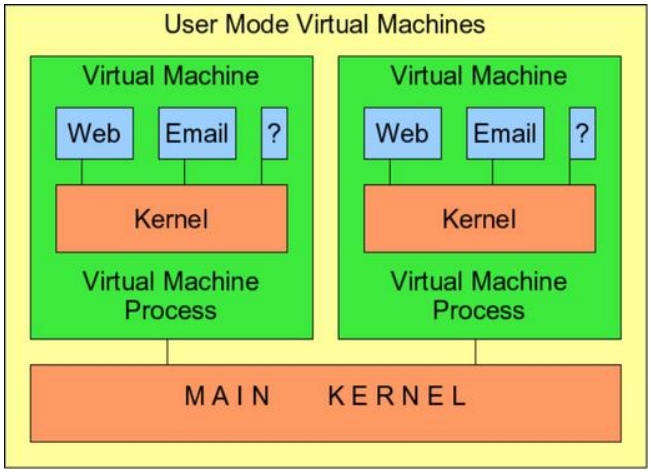
In the "virtual server" model all members run there own service but share the same kernel



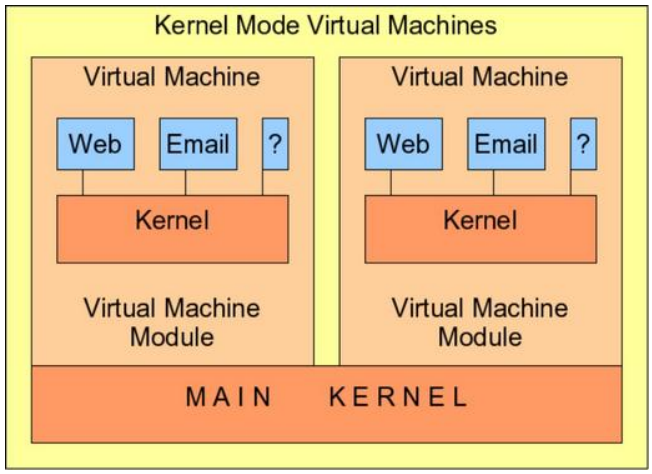
4. Emulated virtual machines

I am thinking specifically about products like QEMU (http://fabrice.bellard.free.fr/qemu/) and Bochs (http://bochs.sourceforge.net). This type of virtualization, or emulation, is unlikely to be used in a web hosting environment, which is the crux of this article, but more in a development type installation where more than one operating system needs to be run. However, I am including it for completeness sake. This operates in a similar way to the "user mode virtual machine" described above except that the "wrapper process" the master machine runs is, in this case, more of a hardware emulator. This allows for more diverse uses of the virtualizing

In the "user mode virtual machine" model all members run there own kernel and services, though in a user process on the "main" machine



In the "kernel mode virtual machine" model, all members also run there own kernel and services, but directly from the "main" kernel itself rather than from a user process



features, such as the ability to run Microsoft's Operating Systems and Linux Systems, in separate virtual machines, simultaneously. The advantages of this is its versatility, the disadvantage is that it suffers a major performance hit in its virtualization, or emulation, process.

5. Kernel mode virtual machines

This is Xen (http://www.cl.cam.ac.uk/Research/SRG/netos/xen/), *and* what I am so excited about. The overall concept is similar to the "user mode virtual machine" described above. However, in this case, the "wrapper process" is not a user process, but is built into the kernel of the master machine. This means that all of the advantages of the "user mode virtual machine" exist, but there is hardly any performance loss from the virtualization process. In short, a process running on a virtual machine runs almost as fast as a process running on the main machine itself. Also, in Xen, it is possible to control further resources on the virtual machines more easily, such as the number of processes to use for each node and so on.

Xen is still in its infancy, even so, distributions are beginning to ship it as standard, and web hosters are beginning to put their toe in the water to see how well it will do (the Xen software that is, not the toe). First impressions are that it is performing well, and I believe it will soon be seen as a standard option for web hosting. This, I believe, will have major implications in the web hosting market. No longer would an owner of a web hosting company choose between "virtual hosts with not much virtualization" and "virtual machines and a major performance hit". The virtual machine no longer has that performance hit! More memory may be required, but that is now a minor investment. This also enables hardware costs to be consolidated into large "big iron" servers. Therefore, virtual machines could outperform dedicated co-located machines by several factors but be supplied at a tiny fraction of the cost. To run that passed you again, currently you can get the functionality of a co-located machine, as well as administration features, for the price of a virtual host. Soon, I believe, you would not only get the functionality and the administration features, but also the performance of one. In fact, it would not surprise me if the norm was for a cheap virtual machine to significantly out perform a far more expensive dedicated co-located machine. I believe the distinction between network and machine will not only be blurred and obfuscated, but will be totally and utterly distorted.

Conclusion

Few of us can justify the cost of having our own machine, even a co-located one. The vast majority of us who have web sites use one kind of virtualization or another. The type most used is the "virtual host". This is the one that most of the web hosters provide, and is the one currently the easi-

est for administrators and end users to administer. However, technology over the internet is exploding. New innovations are being implemented daily – if not hourly! Where they didn't start as free software, versions of them are being written. Often, simply having a "virtual host" is not sufficient to benefit from the innovations. New web hosting companies – as well as some old ones – have smelled the coffee and are providing virtual machines at what would, in the past, be considered as idiotically low prices, often giving functionality and performance far greater than that of an expensive dedicated machine. For those stuck in a "virtual host" who want to make use of this new technology it is time to bite the bullet, grit your teeth, put on your running shoes, pull your socks up, place your inhibitions to one side, charge in and make that leap.

Virtual Machines offer the functionality and more of a co-located machine for the price of a virtual host though at a performance cost

Copyright information

© 2005 by Edward Macnaghten

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at http://www.gnu.org/copyleft/fdl.html

About the author

Edward Macnaghten has been a professional programmer, analyst and consultant for in excess of 20 years. His experiences include manufacturing commercially based software for a number of industries in a variety of different technical environments in Europe, Asia and the USA. He is currently running an IT consultancy specialising in free software solutions based in Cambridge UK. He has a web site (http://eddy.edlsystems.com).

ANSWER SQUAD.com

Tech Questions? We wrote the book!

Feel like roadkill on the Information Superhighway?

Questions about your computer, the Internet, the program you're writing, or even just the commercial application you're stuck working with every day? You've tried searching the Internet for solutions, with mixed results. You've asked people on various mailing lists, just to be flamed or answered with "rtfm!" You've tried tech support at Microsoft, Apple, Adobe or Macromedia, just to be overwhelmed by their cost. And here you are, still puzzled, still stuck, and still having to endure your computing environment rather than enjoy it.

Get roadside assistance with the AnswerSquad!

We're a group of best-selling authors and tech experts who are creating new, innovative and more efficient ways to help you find your computer nirvana: having everything work the way you want, when you want, and how you want.

Visit our website to learn how you can get the answers you need now.

AnswerSquad.com

Xen, the virtual machine monitor

The art of virtualization

Moshe Bar

Virtualization is set to become a key requirement for every server in the data center. This trend is a direct consequence of an industry-wide focus on the need to reduce the Total Cost of Operation (TCO) of enterprise computing infrastructure. In spite of the widespread adoption of relatively cheap, industry standard x86-based servers, enterprises have seen costs and complexity escalate rapidly.

Virtualization is set to become a key requirement for every server in the data center

Today, for every dollar spent on computing hardware, as many as five dollars are spent on lifetime costs – support, maintenance, and software licenses. Operating System Virtualization, a concept pioneered by IBM in 1972 on the System 360, has become a key requirement, because it enables server consolidation, allowing multiple operating system and application images to share each server, cutting both hardware and lifetime costs.

But virtualization offers many, as yet, unrealized benefits – including development, staging and testing, dynamic provisioning, real-time migration, high availability and load balancing. Today's virtualization offerings are crippled by poor performance, lack of scalability, and an inability to offer the fine-grained resource guarantees that are required to provide true application level SLAs, and support dynamic load balancing and high availability. This article introduces Xen, a powerful, free software virtualization technology.

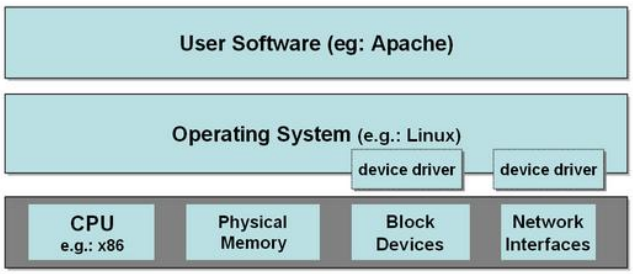
Virtualization: the new infrastructure requirement

The need for Operating System (OS) level virtualization has arisen as a result of a strange coincidence of market forces. First, enterprise software application architectures have become complex, multi-threaded, multi-process and multi-tiered systems, which are difficult to provision, configure and manage.

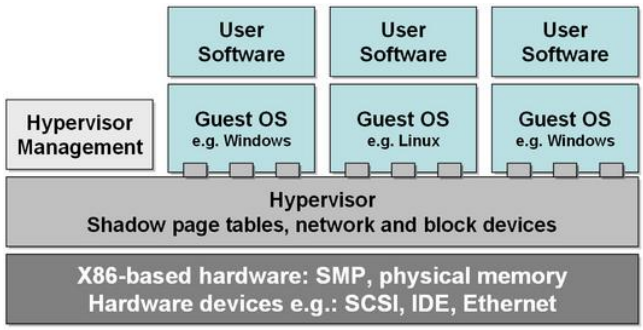
Second, the adoption of so-called "scale-out" computing infrastructure based on inexpensive, industry-standard servers, which has led to a proliferation of servers in the data center.

Frequently, IT staff provision one application per server, because it's the easiest way to ensure that the application and its configuration state can be isolated from other applications in the data center. Moreover, it provides a simple model for dealing with reliability and servicing – if the server fails, only the single application it hosts will fail. If the application must be protected against downtime during server maintenance, or from faults, then it's relatively straightforward to "clone" the entire state of a server, and copy it to an identical machine that can be brought into service to replace the system that goes offline. Finally, provisioning resources at the server level provides a way to identify the true resource needs of an application. If multiple

One App, One Box. On today's servers, one operating system image, together with one application composed of multiple threads and processes, is tied to a single physical server. This leads to higher costs because each physical server requires maintenance and software licenses, and less flexibility because the application load is not matched to the server's capacity, causing over/under utilization



Emulated Virtualization. The guest OS is binary-rewritten to let the hypervisor intercept and manage all changes to hardware data structures, causing frequent address space context switches



applications share a single server it's difficult to determine the real resource needs of each, and to provision additional resources as needed.

Of course, serious drawbacks result from the apparent convenience of tying applications to the physical infrastructure. First, if the application demands less than the full capacity of the server, the CIO will quickly find that most servers are severely under-utilized (typically today, with the incredible capabilities of modern 2- or 4-way servers, utilization figures are about 10-15% per server - Gartner group, August 2004).

Serious drawbacks result from the apparent convenience of tying applications to the physical infrastructure

Of course, each server consumes a full power load, and therefore requires cooling to match. But it also costs about five times as much to maintain – evenly split between the cost of software licenses and the cost of running the server. The net result: proliferation of under-utilized and expensive servers. Finally, the true benefits of scale-out computing are placed firmly out of reach: Easy maintenance, "dial-up/dial-down" provisioning of additional resources in response to the dynamically changing resource requirements of different applications, support for high availability and remote standby and handoff, and an ability to easily develop, test, stage and rapidly provision new applications across distributed data centers are all impossible without the help of OS virtualization.

What virtualization enables

OS virtualization is achieved by inserting a layer of software between the OS and the underlying server hardware. This layer is responsible for allowing multiple OS images (and their running applications) to share the resources of a single server. Each OS believes that it has the resources of the entire machine under its control, but beneath its feet, the virtualization layer transparently ensures that resources are properly shared between different OS images and their applications.

It is important not to confuse OS virtualization with so-called "application virtualization", a software technique that in effect "bundles" all processes, threads and application related state for each different application hosted by an OS, into a virtual container. Application virtualization software vendors, such as Trigence, attempt to provide balanced performance to each virtual container, by applying application-specific policies to the OS scheduler. This achieves few of the benefits of true OS virtualization, the least of which is its inability to take advantage of new hardware features for virtualization, and consequently is not considered a serious contender in the data center.

In OS virtualization, the virtualization layer (often called the hypervisor or Virtual Machine Monitor (VMM)) must manage all hardware structures, such as page tables, and I/O devices, DMA controllers and the like, to ensure that each OS, when running, sees a consistent underlying hardware

layer. Whenever the hypervisor performs a context switch between OS images, it must first preserve any state that the currently running OS will expect to be in place, in the hardware data structures, when its execution is later resumed, and then it must prepare the hardware for the next, incoming OS image. Of course, this comes at a price. The additional overhead that is required to manage all hardware states for the OS, and to present to it an idealized hardware abstraction causes a significant performance overhead. Because many hardware data structures, such as the Translation Lookaside Buffer (TLB), exist to speed up execution within the OS, when these are invalidated on a context switch, performance suffers dramatically because the incoming (newly running) OS image will fault on each page reference until the TLB is refreshed with its state.

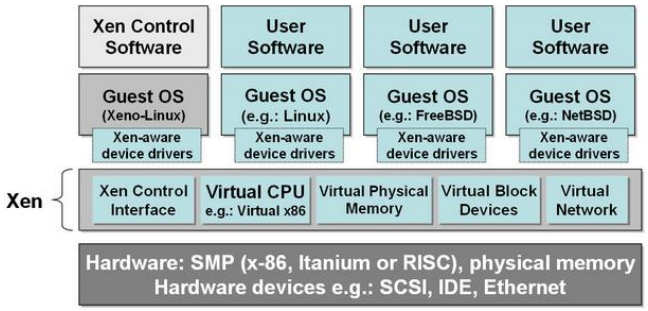
It is important not to confuse OS virtualization with so-called "application virtualization", a software technique that in effect "bundles" all processes, threads and application related state for each different application hosted by an OS, into a virtual container

There is another price too: vendors of virtualization software today charge a hefty premium (multiples of the server cost) for their software, to which must be added the usual OS and application costs. But while today's virtualization products have allowed enterprises to realize significant benefits in the development, testing and QA of n-tier applications, a very high performance hypervisor is a requirement for production-grade server consolidation and to realize the promise of a more dynamic IT infrastructure. Xen, a free software hypervisor, is poised to deliver these benefits, because it outperforms existing hypervisors by an order of magnitude while providing guaranteed service levels to each guest OS. Furthermore, Xen is freely available as free software, and is being broadly supported by major industry players.

Xen: the best in virtualization, for free

Xen uses a very different technique than the hypervisors available today, namely para-virtualization. In para-virtualization, the guest OS is ported to an idealized hardware layer, which completely virtualizes all hardware interfaces. When the OS updates hardware data structures, such as the page table, or initiates a DMA operation, it makes calls into an API that is offered by the hypervisor.

Para-virtualization. The guest OS is ported to the Xen virtual hardware interface. All guest OS modifications of hardware data structures are performed via the API. The hypervisor is mapped into the guest OS address space, avoiding a TLB flush on a context switch into the hypervisor. Guest OSes are optimized for virtualization



Xen fulfills the need for an unencumbered virtualization standard, and offers an opportunity to all players to take advantage of the massive trend towards dynamic datacenter management

This, in turn, allows the hypervisor to keep track of all changes made by the OS, and to optimally decide how to modify the hardware on any context switch. The hypervisor is mapped into the address space of each guest OS, minimizing the context switch time between any OS and the hypervisor. Finally, by co-operatively working with the guest OSes, the hypervisor gains additional insight into the intentions of the OS, and can make the OS aware of the fact that it has been virtualized. This can be a great advantage to the guest OS – for example the hypervisor can tell the guest that real time has passed between its last run, and its present run, permitting it to make smarter re-scheduling decisions to appropriately respond to a rapidly changing environment.

Para-virtualization provides significant benefits in terms of

device drivers and device interfaces. Essentially, device drivers can be virtualized using a para-virtualization model (by splitting the OS drivers into a "top" and "bottom" half), and running the bottom half as a separate domain, with memory, CPU and other resource guarantees. Moreover, the hypervisor itself is protected from bugs and crashes in device drivers, and can make use of any device drivers available on the market. Also, the virtualized OS image is much more portable across hardware, since the low levels of the driver and hardware management are modules that run under control of the hypervisor.

The net result is that Xen offers superb performance – typically more than an order of magnitude faster than any hypervisor on the market. The drawback of para-virtualization is that the guest OS must be ported to the idealized hardware interface. Of course, this is not an issue with operating systems such as Linux, Free BSD, and Solaris. But for closed source operating systems, a para-virtualized hypervisor must rely on hardware support for virtualization to ensure that the native binary of the guest OS can still share resources with other guest OSes.

Xen is a para-virtualizing hypervisor. It relies on one of two approaches to achieve fast virtualization:

1. Hypervisor replicated versions (in memory) of the above state, so that the guest OS is aware it doesn't have full access to and control of the CPU.
2. Hardware based CPU support for multiple guest OSes (replicated stack, task segment structure, GDT and flags) and (in future) support for I/O virtualization.

In the first approach, the hypervisor maintains in-memory copies of all hardware state, and transparently effects changes to the hardware data structures on a context switch, to ensure that the incoming guest OS sees consistent hardware state when it resumes operation. Careful management of state is required to ensure that the minimal set of changes is made to the hardware, to maximize efficiency. This is nowhere more important than in management of virtual memory, via the page table and TLB, by both the hypervisor and the ported OS.

In the second approach, the Xen hypervisor uses hardware based virtualization technologies such as Intel's Silverdale and Vanderpool Technologies (*VT)* or AMD's *Pacifica*. These new capabilities support multiple instances of hardware state, one for each guest OS. Initial versions of this hardware provide CPU support for virtualization, but it is anticipated that in due course these capabilities will be extended into the chipset architectures, to support virtualization of I/O subsystems.

When the major chip vendors ship their CPU support for virtualization, Xen will be able to perform even better. Intel's VT, for example, introduces a software-managed TLB and Global Descriptor Table, which removes the need for Xen to replicate and control these structures, and for the OS to support virtualization of the page tables.

Virtualization and the promise of utility computing

For all the potential benefits of virtualization and utility computing, few enterprises have yet managed to achieve the levels of performance and support for a broad range of software and hardware that they desire. Xen fulfils that need.

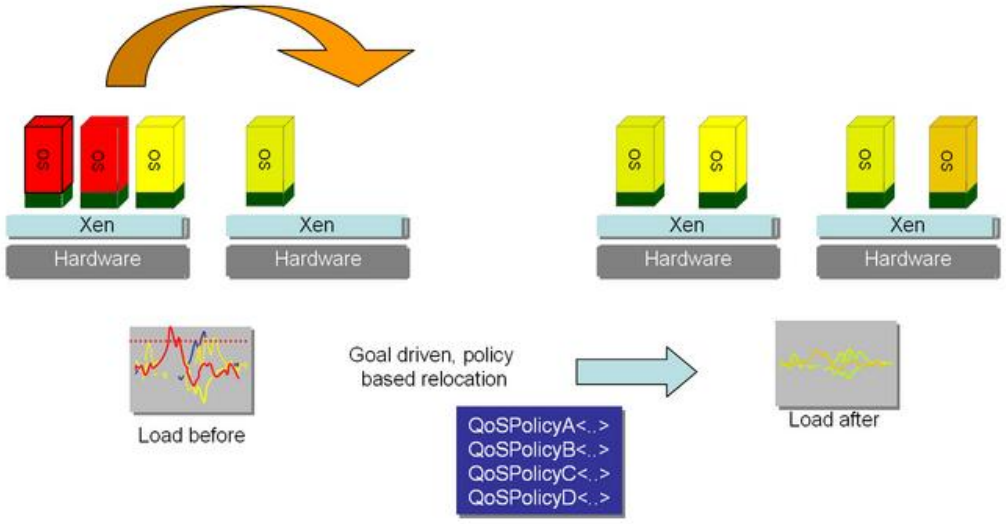
> Xen uses a very different technique than the hypervisors available today, namely para-virtualization

With a high performance hypervisor, it will become possible to deliver on many of the key demands of major enterprises for an adaptive, responsive IT architecture. Xen supports a very wide range of hardware platforms, and therefore its guest OSes can run on a wide variety of hardware. Xen will soon support SMP guest operating systems, a key requirement for applications that today run on large SMP machines. Xen also offers a capability for live VM migration, in which a running guest OS, in its virtual machine, is moved to a second machine in a very short time.

While existing products on the market today claim live migration as a feature, they typically cause the migrated application to be unresponsive for tens of seconds while it is moved. Under Xen, with a feature that enables "copy on write" for guest OS pages, the "downtime" is typically 30-60ms, orders of magnitude faster than available today.

With these raw capabilities, Xen is ideally positioned to allow major enterprises to realize the promise of utility computing. Xen moves the level of infrastructure up above the basic hardware, by providing a common, low-level, high

Xen's Live Relocation. In a data center, Xen's live relocation capability can be used to move a running guest OS and application from one server to another, to achieve dynamic load balancing. This is done while the guest OS is running, with an almost imperceptible interruption in service for the moved image (about 30-60ms)



speed set of execution primitives that can be used to provide a dynamic and responsive computing environment.

The need for a free hypervisor

Today, several hypervisors are available on the market. None are free, and all are closed and tied into expensive, proprietary software stacks. Hardware vendors, rapidly moving to support virtualization, are naturally unhappy at the potential proliferation of virtualization technologies because it has the potential to slow down adoption. To fully take advantage of current and future virtualization features, the best technology should be widely adopted by the market. In addition, major enterprises want a virtualization layer that is not tied to any one OS, and that offers the best performance. Xen fulfills the need for an unencumbered virtualization standard, and offers an opportunity to all players to take advantage of the massive trend towards dynamic datacenter management.

Xen is a free software project, run under the free software community rules. By virtue of its availability, and because it offers the best virtualization technology available, it is a natural candidate for a broadly adopted "standard" hypervisor. The free software community has embraced Xen as offering both the right technology – through its para-virtualization approach and extremely high performance – and lack of bias towards any chip architecture, operating system or application vendor.

Copyright information

(The following license is effective immediately)
© by Moshe Bar
Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

About the author

Free software veteran and openMosix Project leader Moshe Bar is a founder and the CTO of XenSource, Inc. Prior to XenSource, Bar co-founded Qlusters, Inc., where he served as CTO, leading the company's technology and product strategy. Previously, Moshe was VP, ERP implementations, at Baan Europe. He is the author of three books on Linux internals and free software development tools, a senior editor at byte.com, a founding research member of Democritos (the Italian national institute for nuclear simulation), and teaches at the UNESCO and U.N. Atomic Agencies.

Longhorn application development using XAML - learn tomorrow's GUI standard today

formerly Software 2.0
Software Developer's JOURNAL
XAML • Ruby • Java • C/C++ • XML
Software Developer's JOURNAL
new ideas & solutions for professional programmers
5/2005 (5) June 9.90€ ISSN 1734-3933 Monthly
XAML
Rapid application development for Longhorn
HSQLDB
High-performance database engine in Java
Semantic Web Services
Web services speaking natural languages
Redland
C library implementing the RDF Web standard
JRuby
Excellent scripting language coupled with powerful Java libraries
HIT!
ON THE CD
Monoppix 1.0
Bootable Linux for .NET developers
Oxygen 5.1 Trial
Latest version of the full-featured XML editor
Xamlon 1.0 Pro
30-day XAML interpreter trial
WORKSHOP
AspectC++
Crash course in aspect-oriented programming
LIBRARY OF THE MONTH
AnimaDead
Animating 3D characters in C++
SOFTWARE ENGINEERING
The Luminous Ground
Alexander's patterns thirty years later
L 11391 - 5 - F 9,90 €
EUROPE 9.90 € SWITZERLAND 15 FS DOM 11.80 € TOM 1300 XPF
MAROCCO 65 MAD CANADA 17.75 $CAD USA $9.90

Linux-VServer

Resource efficient context isolation

Herbert Pötzl Micah Anderson Björn Steinbrink

Everyone is eager to virtualize their working environment to take advantage of the abstraction layer it provides. Some may require resource isolation for enhanced security, others may need development environments for testing and debugging. Whatever your needs are, virtualization will save you resources through utilizing them more efficiently. This is done by exploiting synergies built on proven technologies, improving availability and reducing downtime, adding scalability through duplication and gaining a certain degree of hardware independence.

Virtualization can be done on different levels, each one with its own advantages and disadvantages

Gains from virtualization

The gains from virtualization are rapidly being uncovered, however the most obvious savings are in maintenance. Maintaining ten virtual instances of a service, application, or system, that are all very similar to each other, is much easier than maintaining ten separate machines, with ten different operating system installations, patch levels, security updates, etc. Keeping all of your virtual instances on one machine is much more resource efficient, and easier to manage.

Different virtualization levels

Virtualization can be done on different levels, each one with its own advantages and disadvantages and each one requiring different implementation techniques. Basically you can virtualize:

- Services (web, mail, ICQ, shell...)
- Applications (desktop, word processing...)
- Userspace (jails, vservers, sandboxes...)
- Hardware (virtual machines, hardware partitions...)

Linux-VServer excels at handling the level of system and application virtualization, by virtualizing exactly those pieces that are required and no more, with as little overhead as possible.

Linux-VServer excels at handling the level of system and application virtualization, by virtualizing exactly those pieces that are required and no more

What "native performance" really means

If we look at virtual machines, whose design includes binary translation or hardware partitioning, to run many instances of different operating systems, or the more recent para-virtualization techniques, like Xen or UML which strive to

Linux-vserver home page

reach "native performance" inside the virtual machine, you might ask, "why is another approach needed?"

Para-virtualization performance measurements are based on a single unit running in a virtual guest environment. As you add more units, more overhead is incurred. The Linux-VServer project is designed to scale virtual units without incurring this additional overhead.

Let's see what this actually means by hypothetically putting each service into its own isolated environment. We'd have a virtual unit for a web server, one for the database server, an FTP server, probably a mail server, a shell server, an IMAP server, maybe even some IRC services, etc. Let's assume we need a dozen different virtual units for our overall "Server" to run.

Reducing the overhead by eliminating the kernel

With Xen or UML you have to provide each unit with a kernel, some memory, disk space, a network, and, of course, some CPU share. This in turn means that you would have about a dozen kernels running, each doing their own file caching, disk buffering, network processing and a bunch of other things that kernels usually do. For example, a syscall to read a file is first processed by the guest kernel, to be then handed upwards and result in an actual I/O by the host kernel, which in turn has to hand back the data to the guest kernel before it reaches the process. Now you might rightfully ask: why would I do that?

- Why add latency and overhead of a dozen running kernels?
- Why buffer and handle the same data many times?
- Why have several network stacks if one is enough?

And this is where Linux-VServer (and, of course, other free and commercial implementations of the same idea) come into play. By virtualizing the interface between processes and the kernel, so that every process (or group of processes) gets a limited view of reality, we can build units very similar to real machines, which can work side by side on the same hardware. Those units can run anything, from a single process to a whole distribution, without the need for a separate kernel, and therefore without the need to process any data twice.

Faster than the real thing?

In a Linux-VServer virtualized environment you don't have a kernel for each instance, but instead the implementation uses contexts and the mostly unknown Linux Capability System to ensure secure interfacing with the kernel. This means that Linux-VServer does not add invisible overhead for each new guest. Instead, you can expect the same performance in a Guest server as compared with the Host server because processes running in the Guest are talking directly to the kernel itself.

Extending the "chroot" concept

The way this is achieved is through context separation and by applying the well-known concept of a "chroot" to a much larger set of resources than is typically done in traditional "jails". Although the Linux-VServer implementation uses the tried and true chroot concept, it is important to note that it also resolves some fundamental flaws in chroot itself, therefore resolving any traditional chroot() escapes. These concepts are then applied to context separation so that process namespace and network addressing can be isolated appropriately. Context separation makes processes have scope that prohibit them from interacting in unwanted ways between processes inside the context and processes belonging to other contexts. This means that in a Guest the groups of processes that run there are isolated from the other Guests on the system, as well as from the host system itself.

To complete the virtual environment several kernel interfaces are modified to return "virtualized" information. Virtualized information allows you to have separate servers whose uptime, the host and domain name, machine type and kernel version are all different in respect to its virtual environment. Similar changes are made for context memory availability and disk space, even on a shared partition.

In addition to that, the administrator of the Host can get a lot of useful information regarding the guest, and in turn control the resources available to each guest, by specifying limits and tuning the scheduler to adjust the process priorities or even stop scheduling processes when the context has used up its CPU share.

Sharing resources by "unification"

Resource sharing is further improved by a concept called "unification" which is based on "protected" hard links, which cannot be altered, but unlinked (to allow updates). Files that are common between different Guests are shared in a manner that does not reduce the level of security of the isolation. Files that are not likely to change, such as libraries or binaries are "unified" so that the amount of disk space, inode caches, and memory mappings for shared libraries is reduced. The Linux-VServer unification process performs the necessary steps to find common files and then hard link them between contexts protecting them against unwanted modification while still allowing them to be removed in the process of updating software inside the Guest.

Resource sharing is further improved by a concept called "unification" which is based on "protected" hard links, which can not be altered, but unlinked (to allow updates)

Hardware independence allows for many platforms

Linux-VServer is fairly hardware independent, which makes it available on basically all known Linux platforms, may it be x86 or x86_64, sparc/64, powerpc/64, mips, alpha or more exotic architectures like sh64, ia64, s390, uml and xen (as soon as it gets into mainline). It is available for 2.4 kernels (with the focus more on stability) as well as recent 2.6 kernels (where new enhancements and features are added).

The current development version contains the following features:

- virtual namespace support (like chroot, but more secure)
- configurable context procfs permissions/visibility
- tagged filesystem support (for shared disk limits)
- modification of utsname information
- resource limits (AS, RSS, NPROC, Files, Locks, IPC, etc.)
- socket, process and memory accounting
- token bucket priority scheduler, hard scheduler

Finally, it should be mentioned that Linux-VServer is a non commercial community project and so you are welcome to join the development or participate in any other way you would like to. For more details have a look here (`http://linux-vserver.org`) or just visit us via IRC on #vserver at irc.oftc.net.

Copyright information

(The following license is effective immediately)

© 2005 by Herbert Pötzl, Micah Anderson, Björn Steinbrink

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at http://www.gnu.org/copyleft/fdl.html

About the author

Herbert Pötzl has studied Computer Sciences and has taught Object Oriented Software Engineering at the Technical University of Vienna. He is currently working as a Consultant for Unix and Linux System Integration and Server Consolidation, and since November 2003 has been the Project Leader for the Linux-VServer Community Project.

Haskell

A very different language

John Goerzen

Many programmers are fluent in several programming languages. Most of these languages have some things in common. Loops and variables are fundamental features of most languages.

I want to show you a different way of solving problems. Haskell takes a different approach than you're used to - to just about everything.

Why Haskell is interesting

There are quite a few things about Haskell that make it interesting and unique:

- Haskell has no loops because it doesn't need them. There is no "for" or "while" in Haskell.
- Haskell has no equivalent of the variables that you're used to; it doesn't need them either.
- Haskell is a functional language. In a language like Java or Python, your primary view of the world is an object. In Haskell, your primary view of the world is a function. I like to say that Haskell manipulates functions with the same ease that Perl manipulates strings. In Haskell, it's commonplace to pass around bits of code. This is a powerful concept.
- Haskell functions are also pure. Every time they're called with the same arguments, they'll return the same result. Functions in most languages can return different results each time they're called. The results may depend on things like a global counter or I/O. Haskell functions also have no side-effects. They won't stomp over a global variable.
- Haskell is a lazy language. It never performs a computation unless it needs to. This is not just an optimization; it's a powerful way to view the world. Code that could be infinite loops or consume vast amounts of memory in other languages are simple, everyday tools in Haskell.
- Haskell can be either interpreted or compiled to native machine code. It also interfaces easily with C. You can call C functions from Haskell with a minimum of hassle. Usually, you'll only need 2 or 3 lines of code to accomplish the call. Haskell also has interfaces to Java, .NET, and Python.
- Haskell lets you write code in a surprisingly intuitive way. Reading Haskell code is easy, and reasoning about Haskell code is easy, too. You'll have less need for a debugger with Haskell.

To get you started, here's an example for a simplistic grep, written in Haskell:

```
import MissingH.List

main = do
   c <- getContents
   putStr (unlines
           (filter (\line -> contains "Haskell" line)
                           (lines c)))
```

This will simply read data from standard input and display all lines containing the word "Haskell" on standard output.

I'll go through this example in more detail and show how it works later in this article.

The Haskell toolbox

To get started with Haskell, you'll need a compiler or interpreter. The most popular compiler is GHC, available from the GHC web site (http://www.haskell.org/ghc/). Some Linux or BSD distributions have GHC packages available; look for a package named `ghc` or `ghc6`. If your operating system doesn't have packages available, sources and binaries for many systems are available from the GHC homepage.

The GHC package actually includes a compiler (ghc) and an interpreter (ghci). Use whichever you like. If you prefer a smaller package that includes only an interpreter, try Hugs also from the HUGS web site (http://www.haskell.org/hugs/). Many distributions also contain Hugs packages.

Haskell can automatically determine the type of a piece of data by looking at how it is created and used

---

Both GHC and Hugs come with a basic library of Haskell code called fptools. A reference is available from GHC's site (http://www.haskell.org/ghc/docs/latest/html/libraries/index.html).

The examples in this article will also use functions from MissingH (http://quux.org/devel/missingh/), a library of useful functions written in Haskell. Many other Haskell libraries are also available for use. See the links at the end of this article for more information.

To compile a Haskell program with `ghc`, you could use a command such as

```
ghc --make -o program program.hs
```

The examples here use MissingH, so you'll need to add

```
-package MissingH
```

at the beginning of your `ghc` command line. You can run Haskell programs with Hugs by saying

```
runhugs program.hs
```

Laziness at work

The grep example at the beginning of this chapter probably doesn't make much sense yet. Here's another version of it that does exactly the same thing, but breaks down the code into more manageable pieces:

```
import MissingH.List

filterfunc line = contains "Haskell" line

main = do
   c <- getContents
   let inputlines = lines c
   let outputlines = filter filterfunc inputlines
   let outputstring = unlines outputlines
   putStr outputstring
```

Let's analyze this version. First, I import the `MissingH.List` module. This module has the `contains` function that I'll be using.

Next, I create a function named `filterfunc`. It takes one parameter, `line`. It calls the `contains` function, passing it two arguments: the string `\Haskell"` and `line`. The `contains` function returns a boolean value (`Bool` type in Haskell). So, `filterfunc` takes a string and returns a `Bool`.

The next line has the `main` function. This is the entry point to the Haskell program, similar to `main()` in C programs. In Haskell, `main` takes nothing and returns an IO action. Actions will be covered in more detail later.

The main function starts by calling `getContents`. This returns the entire contents of standard input as a string. `getContents` is an IO action, so the `<-` operator is used to cause c to represent the result of evaluating the action.

Next, I set up several Haskell variables. The `inputlines` variable holds a list of strings. Each string represents one line from the input. The `lines` function takes a string, separates it by newline characters, and returns a list of the component lines.

The `outputlines` variable also holds a list of strings. It calls `filter` to eliminate all lines that don't contain `\Haskell"`. `filter` is a function that takes a function as an argument. In this case, I pass along `filterfunc`. `filter` returns only those elements from the input list, for which, the passed function returns a `True` value. This model is quite popular in Haskell, and is a very simple illustration of passing functions around.

Then, the `unlines` function is called to combine this list of lines back into a string. Finally, this resulting string is printed.

If this program is looked at from a traditional perspective, it will appear poorly written. It might appear that it starts by reading the entire file into memory - a bad thing if your file is huge. Not so in Haskell.

In Haskell, a string is a list of characters. Because Haskell is lazy, elements of a list are only evaluated when their contents are required for computation. And they can be garbage-collected whenever the compiler knows they won't be needed again. So, when you see `c <- getContents`, nothing actually happens right then.

In fact, nothing at all happens until the very last line in the program. That line demands the content from `outputstring`, which in turn follows up until it reaches `getContents`. It's only now that input is read.

Types and patterns

Haskell is a strongly-typed language like Java or C. However, you probably noticed that I supplied no typing information at all in the grep example. That's because Haskell has another unique feature: type inference. This means that Haskell can automatically determine the type of a piece of data by looking at how it is created and used in a program. Haskell can still catch type errors at compile time, but it saves you from the effort of manually declaring types all the time.

Types can be manually declared for clarity or to make the type more restrictive than the inferred type. Here's an example of the grep program with explicit types given:

```
import MissingH.List

filterfunc :: String -> Bool
filterfunc line = contains "Haskell" line

main :: IO ()
main = do
   c <- getContents
   putStr $ (unlines . filter filterfunc . lines) c
```

The declaration for `filterfunc` says that it takes a `String` and returns a `Bool`. If it took more parameters, you could put more arrows and types in the line; the very last one is the return value.

Types are closely related to patterns. For example, say that you wanted to write your own `filter`. Here's a way it might be done:

```
import MissingH.List

filterfunc :: String -> Bool
filterfunc line = contains "Haskell" line

myfilter :: (a -> Bool) -> [a] -> [a]
myfilter _ [] = []
myfilter f (x:xs) =
  if f x
   then x : myfilter f xs
   else myfilter f xs

main :: IO ()
main = do
   c <- getContents
   putStr $ (unlines . myfilter filterfunc . lines) c
```

The `myfilter` function is the new and interesting one here. Before I discuss how it works, there are several interesting things to note about its type declaration. This function is said to be polymorphic because it works on items of many different types. In this case, it can take a list of any type of item, a function that takes one of those items, and returns a list of the same type of items. The `a` in the type declaration represents this. The first parameter to `myfilter` is given to be a function itself. The second parameter is a list of items, and the return value is a list of the same type of items.

Next, I declare the function itself. The line `myfilter _ [] = []` means that if myfilter is called with an empty list, it returns an empty list. The underscore is a wildcard and means that it doesn't matter what function is supplied. In fact, `_ []` is a simple instance of pattern matching in Haskell.

The next line contains `myfilter f (x:xs)`. In Haskell, the colon represents the list you get by adding a single item to the beginning of the list. So, this pattern will put the first item of the list into `x`, and the rest of the list into `xs`. Note that `xs` may be empty if the list has only one item.

Now, the passed function is called, passing in the current item. If the function returns `True`, the return value can be thought of as being the current item plus the result of filtering the rest of the list. Hence the line containing `x : myfilter x xs`. This becomes the return value; the function calls itself. This is recursion, and is the most common way to achieve in Haskell what would be looping in other languages.

You can also define your own data types in Haskell. Here's an example:

```
data Maybe a = Nothing | Just a
```

This defines a new polymorphic type, `Maybe a`. A value of type `Maybe a` can be created in two ways. First, you could simply say `Nothing`. Secondly, you could say `Just x`, where `x` is some value of type `a`. Pattern matching works just as well with custom types as it does with built-in types.

The `Maybe` type is, in fact, such a useful pattern in Haskell that it is defined for you in the Haskell `Prelude` - the set of functions and types available to every Haskell program. Functions that may either compute a value or generate an error frequently use `Nothing` to indicate a problem, or `Just x` to indicate a successful calculation.

Functions

A little bit of the versatility of Haskell's functions has been seen already, when a function was passed to a filter. I'll now look at some other things you can do with functions.

In the first grep example, there was this line: `\line -> contains \Haskell" line`. It declared a new function on the spot. The backslash begins a declaration. The function took one parameter (`line`), and calculated its result by applying the part on the right. Functions declared like this are often called anonymous functions because they are never bound to a name.

As you've probably noticed, to call a function, you list its name and all parameters to it, separated by a space. There is a unique twist to that. The `contains` function is defined in MissingH with this type:

```
contains :: [a] -> [a] -> Bool
```

Since a `String` is a list of `Chars` in Haskell, this works well for filtering `Strings`.

Say `contains` is called with only one argument. In most languages, that will generate an error. In Haskell, however, it returns a new function, with the leading arguments no longer needing to be specified. This is called partial application. So, the type of `contains \Haskell"` is `String -> Bool`. Note that the type isn't `[a] -> Bool`. Because the first argument was given as a `String`, we know the next argument must also be a `String`.

So, instead of saying `\line -> contains \Haskell" line`, I could have said simply `contains \Haskell"`.

Did you notice the last line of the last grep example looked unusual? That line was:

```
putStr $ (unlines . myfilter filterfunc . lines) c
```

The period is a function composition operator. In general terms, where `f` and `g` are functions, `(f . g) x` means the same as `f (g x)`. In other words, the period is used to take the result from the function on the right, feed it as a parameter to the function on the left, and return a new function that represents this computation. The dollar sign is a bit of syntactic sugar that simply removed the need to put everything after `putStr` in parenthesis.

Variables

Recall that I said that Haskell has no variables in the conventional sense. You might be wondering about the let statements in the second grep example. Haskell does have "variables", and `let` is one way to declare them. A Haskell variable doesn't hold a value and can't be modified. Instead, a Haskell variable tells the compiler, "if you ever need to know the value of x, here's how you calculate it." Assigning something to a variable doesn't cause it to be calculated; in fact, if the value is never needed, it will never be calculated. Thus, a variable in Haskell is just a shortcut, similar to a macro in some other languages.

Monads and I/O

You've seen a very small bit of the power of functions so far. Monads are used to combine functions together in a way similar to the period operator, feeding the result of one to the input of the next. However, monads provide more capabilities. For instance, a monad can abort the processing of an entire chain when there is a problem anywhere along it. The `Maybe` monad, for instance, can receive `Just 5` from one function, pass `5` to the next, receive `Just 6` from it, pass `6` to the next, and continue doing that across many functions. If any function returns `Nothing`, the computations stop, and the result of the entire computation becomes `Nothing`. Otherwise, the result of the entire computation is the result of the last function in the chain.

I/O was historically a tricky problem for pure languages like Haskell. A function that reads data from the keyboard obviously can't be guaranteed to return the same thing each time it is invoked.

In Haskell, the `IO` monad is used. The `IO` type is opaque, meaning that a Haskell program can't see "inside" it. By using constructs like `<-`, however, things can be read and written. The `<-` operator extracts the value from the inside of a monad type and assigns it to a variable. If you were using the `Maybe` monad and wrote `x <- Just 5`, then `x` would evaluate to `5`.

The `IO` monad is inescapable, however. Once you call `IO` functions, your return value will be in the `IO` monad. That is, your return type might be `IO Int` or `IO String`. This provides a neat way of segmenting impurities.

Typically, Haskell programs are structured so that the outermost layers are in the `IO` monad, and computations are outside of it. The main function returns `IO ()` - an empty value in the `IO` monad. So, to execute a Haskell program, the compiler simply evaluates the I/O action that main represents, calling other functions as needed along the way.

Typeclasses: OOP in reverse

Object-oriented programming (OOP) is a fixture of many languages. OOP, in general, permits you to write code that accepts an object or any child of that object. It's a way to conceptualize the view of the world.

Haskell provides something similar called typeclasses. Typeclasses let your functions take data of any type, so long as a particular interface for that type exists. Instead of preventing access to the internal representation of data in an object, typeclasses, Haskell provides a way to handle many different types of data in a generic way.

For instance, there is a built-in function called `show`. The `show` function can generate a string representation from many different data types. Its type is this:

```
show :: Show a => a -> String
```

This can be read as "The show function takes any value of type `a`, such that `a` is part of the typeclass `Show`, and returns a `String`." You can say `show \Hi"`, or `show 5.0`, or even `show True`, and get a valid `String`.

You can add your own data types to the Show typeclass very easily:

```
data MyType = Red | Blue
instance Show MyType where
  show Red = "Red"
  show Blue = "Blue"
```

The `Show` class itself could be defined like this in the Prelude:

```
class Show a where
  showsPrec :: Int -> a -> ShowS
  show    :: a -> String
  showList :: [a] -> ShowS

  showsPrec _ x s = show x ++ s
  show x       = showsPrec 0 x ""
  showList = \ldots{}
```

Here, it can be seen that to be an instance of `Show`, normally three functions would have to be provided. However, in this case, defaults are provided, so really, only one function is required.

Instead of preventing access to the internal representation of data in an object, typeclasses, Haskell provides a way to handle many different types of data in a generic way

Typeclasses are powerful abstractions in Haskell. The `Num` typeclass, for instance, is used to provide an abstraction of arithmetic operators. The type of (+), the function representing the + operator, is `Num a => a -> a -> a`. Numeric types are all instances of`Num`, and thus + can be used with many different types of numbers. You can invent your own numeric types and, by simply making them instances of `Num`, all existing numeric operators will work with them.

Conclusion

Haskell is a powerful and flexible language. Its approach to solving problems is unique and refreshing. The ability to combine functions is powerful and time-saving. There is a great deal of power in Haskell, which is easily tapped, but a magazine article such as this can barely scratch the surface. I encourage you to seek out more detailed resources about Haskell.

Here are some resources for more information on Haskell.

For general information, look at:

- The Haskell home page haskell.org (http://www.haskell.org)
- The Haskell wiki haskell.org/hawiki (http://www.haskell.org/hawiki)

Tutorials and references:

- Haskell online resources (http://www.haskell.org/learning.html)
- Yet Another Haskell Tutorial, in my opinion the best Haskell tutorial available (http://www.isi.edu/~hdaume/htut/)

Libraries and code:

- Haskell at Freshmeat freshmeat.net/browse/834/ (http://freshmeat.net/browse/834/)
- Libraries in Haskell haskell.org/libraries (http://haskell.org/libraries)
- Applications in Haskell haskell.org/practice.html (http://haskell.org/practice.html)
- Libraries wiki page haskell.org/hawiki/LibrariesAndTools (http://haskell.org/hawiki/LibrariesAndTools)

Copyright information

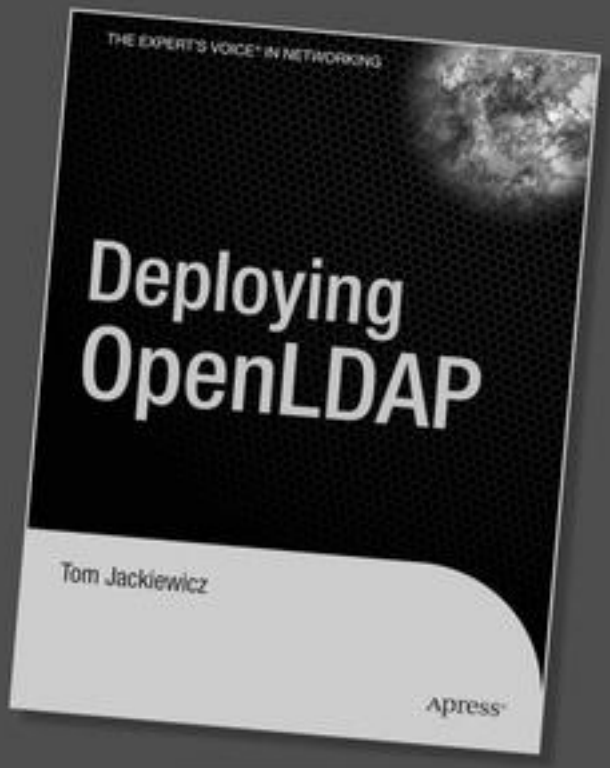
(The following license is effective immediately)
© 2005 by John Goerzen
This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

John Goerzen is an avid programmer, developer for the Debian GNU/Linux project, and a systems administrator. He is the author of several books, including the recent Foundations of Python Network Programming. John is currently the president of Software in the Public Interest, Inc., the legal parent organization of Debian.

YOU NEED TO KNOW MORE ABOUT CREATING NEW DIRECTORIES

Check out *Deploying OpenLDAP* by Tom Jackiewicz!



Published November 2004
ISBN: 1-59059-413-4 • $39.99 US • 344 pages

"This is an excellent introduction to OpenLDAP."
—Jack Herrington, Code Generation Network

Deploying OpenLDAP delves into the logic, theories, and fundamentals of directories. Focusing on open standards, this practical book shows you how to design and deploy the most useful directory possible.

For more information about *Deploying OpenLDAP* or other Apress titles, please visit **www.apress.com**. Apress books are available at fine bookstores worldwide.

Apress®

Newsflash!

RSS: beyond news sites and weblogs

Saqib Ali

RSS (Really Simple Syndication) is an XML based web content syndication format. RSS has become the defacto feature on weblogs and many news sites. Almost all major news sites and weblogs provide an RSS feed for their audience. An RSS-aware program (aka RSS reader) can check these RSS feeds for changes and display the updates in a human readable format.

Almost every computer geek visits Slashdot.org once a day. But UberGeeks, like me, prefer to be always up to date with the latest articles on Slashdot.org. So instead of visiting Slashdot.org every 5 minutes, I have subscribed to the Slashdot RSS feed. As soon as there is a new article on Slashdot, my RSS reader notifies me of it. This allows me to logon and make the "First Post" (reply) to the Slashdot.org article. I have to attribute many of my "Slashdot First Posts" to the power of RSS. This power of RSS has been utilized on other news sites as well. NPR, CNN, and Wired all provide RSS feeds.

URLs to RSS feeds from some popular weblogs and news sites:

- Slashdot (http://slashdot.org/index.rss)
- National Public Radio (http://www.npr.org/rss/index.html)

However there are other areas where the power of RSS has not been fully realized. Wikis, Usenet and web based discussion groups come to mind. But this is changing fast. In this article I would like to go over some of the free software that allows web-content distribution and republication in the areas of Wikis, Usenet and discussion groups. I'll also go over a powerful RSS Reader that is freely available.

Firefox: a powerful RSS reader

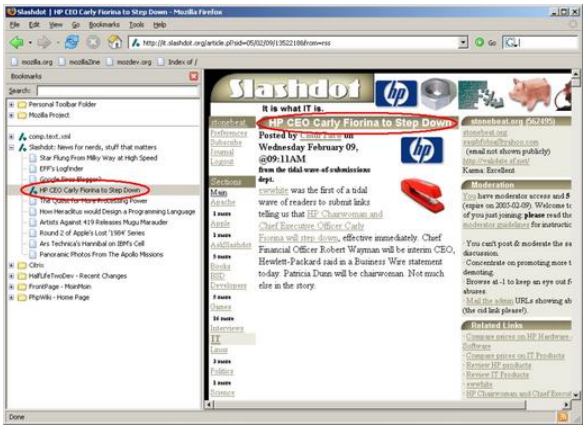
An RSS Reader is an application that polls RSS feeds and displays them in a human-readable format. The reader allows you to browse the newly available items in the RSS feed. RSS readers come in many flavors.

One powerful RSS reader, that often goes unused by many, comes built-in with Mozilla Firefox browser. It is called the *Live Bookmark*. Live Bookmarks is a new technology in Firefox that lets you view RSS news and weblog headlines in the bookmarks toolbar or bookmarks menu. It enables you to quickly see the latest headlines from your favorite sites. Clicking on any of the live bookmark will take you directly to the page referenced by that RSS item.

You can download Firefox from the Mozilla web site (http://www.mozilla.org/products/firefox/). Some other Freely available RSS readers:

- RSS OWL (Java based) (http://www.rssowl.org/)
- FeedReader (Windows) (http://www.feedreader.com/)
- DSSBandit (windows, written in C#) (http://www.rssbandit.org/)
- LifeArea (GTK/GNOME) (http://liferea.sourceforge.net/)

Firefox's Live Bookmark lets you see the latest headlines from your favorite sites. Clicking on any Live Bookmark opens up the full article in the browser window



- Straw (Python/GNOME) (http://www.nongnu.org/straw/)
- Syndigator (Perl/GTK) (http://syndigator.sourceforge.net/)
- Blam (Mono: C# and Gtk#) (http://www.imendio.com/projects/blam/)
- Snownews (Console based, ncurses) (http://kiza.kcore.de/software/snownews/)

Chronological web sites?

Wiki is a web site that allows users to add, edit or modify web site content, by merely using a web browser. Wiki provides a fast and easy way to collaborate and collaboratively create documentation on the web. Not surprisingly, Wiki comes from the Hawaiian term for "quick" or "super-fast". A Wiki engine also serves well as a Departmental or a Taskforce web site, which the team members can use to share ideas and publish content, without messing around with HTML editors. PHPWiki, as the name suggest, is a PHP based Wiki, while Twiki is PERL based. Both of these Wiki engines, with their aim to foster information flow among the users, provide RSS feeds of the latest content and updates in *chronological order*. The RSS feeds from these Wiki engines provide an easy way for the users to keep up to date on the latest updates to the content of the web site. This is especially useful when the Wiki is being used to track bugs or features in a product. The subscriber of the RSS feed will be notified as soon as an update is made to the list of bugs or the feature.

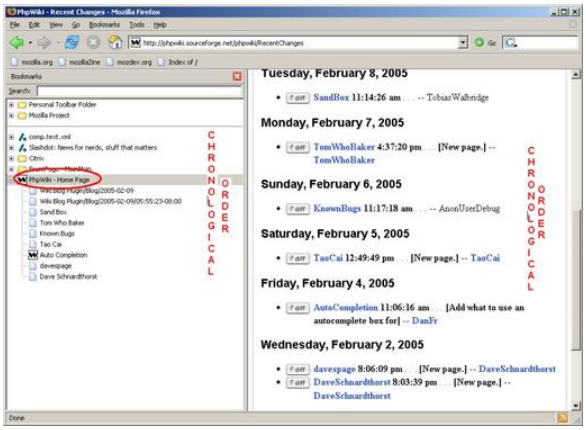
- Twiki (requires Apache and Perl) (http://twiki.org/)
- PmWiki (requires Apache and PHP (http://www.pmichaud.com/wiki/PmWiki/PmWiki)
- MediaWiki (requires Apache, PHP and mySQL) (http://wikipedia.sourceforge.net/)
- MoinMoin (requires Apache and Python) (http://moinmoin.wikiwikiweb.de)
- usemod (requires Apache and PERL) (http://www.usemod.com/cgi-bin/wiki.pl)
- PHPwiki (requires Apache, PHP and mySQL) (http://phpwiki.sourceforge.net/phpwiki/)
- ChiqChaq (requires Apache and PERL) (http://chiqchaq.sourceforge.net/)

Die hard usenet fan?

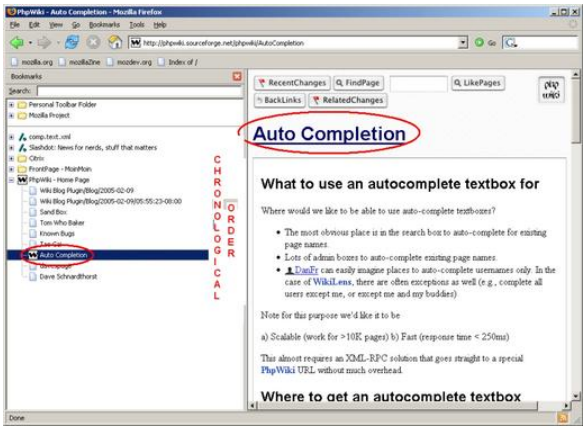
If you are die-hard fan of Usenet, like me, you can never stop pressing the refreshing key to retrieve the latest newsgroup postings. Refresh button (F5), on my keyboard, has worn out due to overuse. Worry not, Google + RSS has come to our rescue. A few years back Google acquired Deja News, the largest collection of web accessible Usenets.

PHPWiki and Twiki are two popular Wiki engines that are freely available under the GNU Public License. Both of these Wiki engines, in their aim to foster information flow

All of the changes made to the content on the Wiki are shown as RSS items in the order they were made (chronological)



Clicking on any Wiki Page Title (RSS item), will take you directly to the modified page. Only the pages that changed/edited show up as RSS items



among the users, provide RSS feeds of the latest content and updates in chronological order

After the acquisition, Google made some very cool enhancements to the Deja News Network. But that wasn't good enough, Google went on to develop a whole web based collaboration platform based on the NNTP protocol. A powerful feature of this collaboration platform is ability to generate RSS feed of the latest newsgroup postings in a chronological order. This allows an avid Usenet fan to keep up to date with the newest post on the groups of interest. As soon as a new Usenet posting is made, the RSS reader on my desktop notifies me of it.

You can subscribe to the RSS feed of a Usenet newsgroup by simply adding the following URL to your favorite RSS reader: `http://groups-beta.google.com/group/{name.of.newsgroup}/feed/msgs.xml`. Replace `{name.of.newsgroup}` with newsgroup of your choice. Some examples:

- http://groups-beta.google.com/group/comp.text.xml/feed/msgs.xml
- http://groups-beta.google.com/group/microsoft.public.visio/feed/msgs.xml

Do you Yahoo?

Ok, so you are not an old timer like me. Instead, you are from the new generation and instead use Yahoo Group to communicate with your peers. Worry not, Yahoo Group also provides a RSS for the groups.

You can use the following URL syntax to subscribe to an RSS feed of the Yahoo Group: `http://groups.yahoo.com/group/{group_name}/rss`. Replace `{group_name}` with the Yahoo Group of your choice. Some examples:

- http://rss.groups.yahoo.com/group/apache-user-group/rss
- http://rss.groups.yahoo.com/group/citrix/rss
- http://rss.groups.yahoo.com/group/ssl-talk/rss
- http://rss.groups.yahoo.com/group/xml-doc/rss

Conclusion

RSS is a fast and powerful way to get news out to an audience. RSS readers inform their users of any new postings on news sites, newsgroups and weblogs. If you need to keep up to date on the latest news articles, or you need to track the postings on Wiki or your favorite weblog, then you really need to use an RSS reader.

Copyright information

(The following license is effective immediately)

© 2005 by Saqib Ali

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at http://www.gnu.org/copyleft/fdl.html

About the author

Saqib Ali is a Sr. Systems Administrator and Technology Evangelist at Seagate Technology. He also manages a free software web based application (`http://validate.sf.net`) that allows online conversion of DocBook XML to HTML or PDF. Saqib is also an active contributor to The Linux Documentation Project

Web site blocking techniques

How to use Squid and squidGuard to restrict user access to undesirable web sites

Tedi Heriyanto

For a variety of reasons, organizations have very strict policies regarding web site access. These policies usually mean that not all users have permission to access all web sites.

This article will explain two techniques that can be used to block web site access to specified groups of users at specified times, using Squid's built-in mechanism and using squidGuard.

In this article, the configurations shown are taken from real files that are used by my clients. I have attempted to write the configuration and installation procedures so that they will work with any operating system. However, where there are specific procedures, I only explain how to do things using an RPM-based Linux distribution.

In the last section, I list advantages and disadvantages using each technique. I also mention common problems that you may encounter during the installation and configuration phase.

Introduction

During my work as a Linux consultant, I'm often asked to implement a mechanism that will stop internet users from accessing inappropriate sites, such as sites containing porn and other offensive or inappropriate material.

There are various reasons organizations might want such mechanisms implemented. The main reasons are:

- **Limited bandwidth** - Some of my clients have very limited bandwidth; they are usually connected to the internet using a dial up modem or a leased line, which allow very limited bandwidth (between 56Kbs and 128 Kbps). In such situations, management cannot permit employees to download inappropriate material as it uses up precious bandwidth.
- **Organizational policy** – Many organizations have very strict internet policies regarding offensive material. For this and other reasons, they don't want employees gaining access to inappropriate sites.
- **Working hours** – Many organizations don't want employees to access particular sites during certain hours.

> Organizations implement web site blocking policy because of limited bandwidth, organizational policy or working hours

To implement this mechanism, I usually use Squid and squidGuard.

From an ethical point of view, before implementing this kind of policy, the users must be informed about the company's policies - it's even better if they are involved in the policy making process. The organization's internet users, management, and IT (Information Technology) department must define what kind of policy will be implemented.

Before implementing web site blocking policy, you have to ensure it conforms with any legislation.

Testing the Squid blocking mechanism by accessing oracle.com

ERROR: The requested URL could not be retrieved - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Index of /pub/linux
ERROR: The requested U...
ERROR
The requested URL could not be retrieved
While trying to retrieve the URL: http://www.oracle.com/
The following error was encountered:
• Access Denied.
Access control configuration prevents your request from being allowed at this time. Please contact your service provider if you feel this is incorrect.
Your cache administrator is
Generated Tue, 01 Feb 2005 06:45:32 GMT by (squid/2.5.STABLE5)
Find: Find Next Find Previous Highlight Match case

Squid is a high-performance, proxy-caching server for web clients, supporting FTP, gopher, and HTTP data objects. Squid can also be used to implement access control.

SquidGuard (http://www.squidguard.org) is a fast and free filter, redirector and access controller for Squid. It was written by Pål Baltzersen and Lars Erik Håland.

In order to use Squid's built-in blocking mechanism, you don't need squidGuard, but you do need Squid to use squidGuard.

Squid's built-in blocking mechanism

In my experience, Squid's built-in blocking mechanism or access control is the easiest method to use for implementing web site blocking policy. All you need to do is modify the Squid configuration file.

Before you can implement web site blocking policy, you have to make sure that you have already installed Squid and that it works. You can consult the Squid web site (http://www.squid-cache.org) to get the latest version of Squid and a guide for installng it.

To deploy the web-site blocking mechanism in Squid, add the following entries to your Squid configuration file (in my system, it's called `squid.conf` and it's located in the `/etc/squid` directory):

```
acl bad url_regex "/etc/squid/squid-block.acl"
http_access deny bad
```

The file `/etc/squid/squid-block.acl` contains web sites or words you want to block. You can name the file whatever you like. If a site has the URL or word listed in `squid-block.acl` file, it won't be accesible to your users. The entries below are found in `squid-block.acl` file used by my clients:

```
.oracle.com
.playboy.com.br
sex
...
```

With the `squid-block.acl` file in action, internet users cannot access the following sites:

- Sites that have addresses ending with .oracle.com
- Sites that have addresses ending with .playboy.com.br
- Sites containing the word "sex" in its pages

In order to use Squid's built-in blocking mechanism, you don't need squidGuard, but you do need Squid to use squidGuard

You should beware that by blocking sites containing the word "sex", you will also block sites such as Middlesex University, Sussex University, etc. To resolve this problem, you can put those sites in a special file called `squid-noblock.acl`:

```
^http://www.middlesex.ac.uk
^http://www.sussex.ac.uk
```

You must also put the "no-block" rule before the "block" rule in the Squid configuration file:

```
...
acl special_urls url_regex "/etc/squid/squid-noblock.acl"
http_access allow admin_ips special_urls

acl bad url_regex "/etc/squid/squid-block.acl"
http_access deny bad
...
```

After editing the ACL files (`squid-block.acl` and `squid-noblock.acl`), you need to restart Squid. If you install the RPM version, usually there is a script in the `/etc/rc.d/init.d` directory to help you manage Squid:

```
# /etc/rc.d/init.d/squid reload
```

To test to see if your Squid blocking mechanism has worked, you can use your browser. Just enter a site whose address is listed on the `squid-block.acl` file in the URL address. In the example above, I block `.oracle.com`, and when I try to access oracle.com, the browser returns an error page.

Using squidGuard blocking mechanism

SquidGuard is:

- Free
- Very flexible
- Very fast
- Easy to install; and
- Portable

You can use squidGuard to:

- Limit user web access to a particular web server or URL
- Stop some users from accessing forbidden web servers or URLs
- Block access to URLs matching a regular expression (list or word)
- Forbid the use of IP addresses in URLs
- Redirect blocked URLs to a smart CGI page
- Redirect popular sites to a local mirror
- Have different rules for different times; and
- Have different rules for different user groups

In this article, I will only use squidGuard to block users accessing some sites. Here is how you install squidGuard:

- Install BerkeleyDB version 3.2.9
- Install squidGuard
- Create a squidGuard configuration file that suits your needs
- Create the domain, URL and expression lists you want
- Test squidGuard
- Configure Squid to use squidGuard as the redirector
- Reload Squid

I will describe this procedure in the following sections.

The output of the command "rpm -qa | grep db"

tedi@suse:~ - Shell - Konsole
tedi@suse:~> rpm -qa | grep db
db-4.2.52-90
gdb-6.2.1-2
gdbm-1.8.3-229
tedi@suse:~> rpm -qi db
Name : db Relocations: /usr
Version : 4.2.52 Vendor: SUSE LINUX AG, Nuernberg, Germany
Release : 90 Build Date: Tue 05 Oct 2004 05:32:41 PM WIT
Install date: Sat 18 Dec 2004 05:44:48 PM WIT Build Host: g190.suse.de
Group : System/Libraries Source RPM: db-4.2.52-90.src.rpm
Size : 4044910 License: BSD
Signature : DSA/SHA1, Tue 05 Oct 2004 05:36:35 PM WIT, Key ID a04edac09c800aca
Packager : http://www.suse.de/feedback
URL : http://www.sleepycat.com
Summary : Berkeley DB Database Library
Description :
The Berkeley DB Database is a programmatic toolkit that provides
database support for applications.
This package contains the necessary runtime libraries.
Distribution: SuSE Linux 9.2 (i686)
tedi@suse:~>

BerkeleyDB and squidGuard installation

Before you can install squidGuard, make sure your system already has BerkeleyDB. You have to match the BerkeleyDB version with the squidGuard version for your system; for example squidGuard version 1.2.0-2 for Red Hat Enterprise Linux 3 needs BerkeleyDB version 4.1.

For RPM-based systems, you can do the following to check if your system has BerkeleyDB or not:

```
$ rpm -qa | grep db
```

If your system doesn't have BerkeleyDB, you must install it first. Here is the command to install the RPM version:

```
# rpm -ivh db-x.y.z.*.rpm
```

In my experience, the easiest method for installing squidGuard is by using the binary RPM version. A binary, RPM version of squidGuard for a Red Hat based system can be found on Dag Wieers web site (http://dag.wieers.com/packages/squidguard/)

SquidGuard configuration and database creation

Before you can use squidGuard, you have to create the squidGuard configuration file and database. You should start the configuration file with only the configuration you need and then extend it later as required.

In my SuSE system, the default path for squidGuard Dag Wieers' RPM version is `/etc/squid/`

`squidguard.conf`. In other distributions, the configuration file could be `/etc/squid/squidGuard.conf` or `/etc/squidGuard.conf`. The default `squidguard.conf` included with the RPM version can't be used without modification.
Usually my squidGuard configuration file has a structure like this:

- Path declaration
- Source group
- Destination group
- Access control rules

I will only describe the structure used by my configuration. If you want to know more about the squidGuard configuration, please see the file included in the squidGuard package (*configuration.html* or *configuration.txt*, in my system it is in `/usr/share/doc/packages/squidGuard-1.2.0` directory).

The organization's internet users, management, and IT (Information Technology) department must define what policy the mechanism will implemement

Path Declaration

The path declaration defines the directory for the log files and the database list. For example:

```
dbhome /var/lib/squidGuard/blacklists
logdir /var/log/squidGuard
```

This declaration will set:

- The directory for the squidGuard database list to `/var/lib/squidGuard/blacklists`
- The directory in which to store the log files to `/var/log/squidGuard`

Source Group

Source group has a general syntax like this:

```
src | source name {
    specification
    specification
    ...
}
```

Specification can be any reasonable combination of:
IP addresses and/or ranges (multiple)

- ip xxx.xxx.xxx.xxx [...]
- ip xxx.xxx.xxx.xxx/nn [...]
- ip xxx.xxx.xxx.xxx/mmm.mmm.mmm.mmm [...]
- ip xxx.xxx.xxx.xxx-yyy.yyy.yyy.yyy [...]

Where:

- xxx.xxx.xxx.xxx is an IP address (host or net, i.e. 10.11.12.13 or 10.11.12.0)
- nn is a net prefix (i.e. /23)
- mmm.mmm.mmm.mmm is a netmask (i.e. 255.255.254.0) and
- yyy.yyy.yyy.yyy is a host address (must be >= xxx.xxx.xxx.xxx)

IP address/range list (single): iplist filename
where:

- filename is either a path relative to dbhome or an absolute path to a database file
- the iplist file format is simply addresses and/or networks separated by a newline as above (without the IP keyword).

For example, the following snippet is in one of my client's squidGuard configuration file:

```
#
# SOURCE ADDRESSES:
#
src admin {
ip       10.10.0.10
}
src users {
    ip       10.10.0.0/255.255.0.0
    ip       10.11.0.0/255.255.0.0
    ip       10.12.0.0/255.255.0.0
}
```

I define two source groups, the *admin group*, which has the IP address 10.10.0.10; and the *users groups*, which have IP addresses in subnetwork of 10.10.0.0, 10.11.0.0 and 10.12.0.0.

Destination Group

Destination group has the following syntax:

```
dest | destination name {
specification
specification
...
}
```

Specification can be any combination of zero or one of each of:

- Domainlist (single):

```
domainlist filename
```

- URL list (single):

```
urllist filename
```

Filename is a text-based database file. You can define the file relative to the database directory path, or you can also define the file absolutely.
Below are the destination groups definitions from my client's configuration file:

```
#
# DESTINATION CLASSES:
#
dest gambling{
log gambling
domainlist gambling/domains
urllist gambling/urls
redirect http://localhost
}
dest warez{
log       warez
domainlist  warez/domains
urllist     warez/urls
redirect http://localhost
}
dest porn{
log       porn
domainlist porn/domains
urllist     porn/urls
redirect http://localhost
}
```

In the configuration above, I define three destination groups: *gambling*, *warez* and *porn*. Their domains and URLs are listed in files called `domains` and `urls` in the directories `gambling`, `warez` and `porn` located in `/var/lib/squidGuard/blacklists/`. When a user tries to access domains and URLs listed in the database, they are redirected to http://localhost.

Access Control List Rules

The Access Control List (ACL) combines the previous definitions into distinct rulesets for each destination and source group:

```
acl {
sourcegroupname {
pass [!]destgroupname [...]
[redirect [301:|302:]new_url]
}
  ...
  default {
pass [!]destgroupname [...]
redirect [301:|302:]new_url
}
}
```

Below is an ACL example:

```
#
# Access Control Lists
#
acl {
admin {
        pass  all
}
users {
pass !gambling !warez !porn all
     redirect 302:http://localhost
}
default {
pass !porn all
     redirect 302:http://localhost
}
}
```

In this configuration:

- The administrators computer has access to every site.
- The user's computers are blocked from gambling, warez and porn domains and URLs listed in the database. When a user tries to access forbidden sites, he or she is redirected to http://localhost
- Computers not listed in the source group are not allowed to access porn domains and URLs. Again, they will be redirected to http://localhost when access porn domains and URLs

SquidGuard database

SquidGuard uses a database that can be divided into an unlimited number of distinct categories like "gambling", "warez", "porn" etc. Each category may consist of separate unlimited lists of domains, and URLs. You can download the blacklists database from here (http://ftp.teledanmark.no/pub/www/proxy/squidguard/contrib/blacklists.tar.gz).

Domainlists

The domainlist file has a simple format:

```
domain
domain
...
```

As an example, for a porn category:

```
playboy.com
```

SquidGuard will match any URL with the domain name itself and any sub-domains and hosts (i.e. playboy.com, www.playboy.com, whatever.playboy.com and www.what.ever.playboy.com but not `.*[^.]playboy.com` (i.e. pplayboy.com etc.)).

URLlists

The urllist file has this format:

```
URL
URL
...
```

with the `proto://((www|web|ftp)[0-9]*)?` and `(:port)?` parts and normally also the ending `(/|/[^/]+\.[^/]+)$` part (i.e. ending `/` or `/filename`) taken out (i.e. `http://www3.foo.bar.com:8080/what/ever/index.html` equals `foo.bar.com/what/ever`)

For instance a category for banned sites could be:

```
foo.com/~badguy
bar.com/whatever/suspect
```

All these URLs will match the above urllist:

```
http://foo.com/~badguy
http://foo.com/~badguy/whatever
ftp://foo.com/~badguy/whatever
http://www2.foo.com/~badguy/whatever
http://web56.foo.com/~badguy/whatever
```

but not:

```
http://barfoo.com/~badguy
http://bar.foo.com/~badguy
http://foo.com/~goodguy
```

Configuring Squid to use squidGuard

Because squidGuard is a redirector for Squid, it needs to be called from Squid. Add the following line to the Squid configuration file (`squid.conf`) to instruct Squid to use squidGuard as a redirector:

```
redirect_program /usr/bin/squidGuard -c
  /etc/squidguard.conf
```

Please change `/usr/bin/squidGuard` and `/etc/squidguard.conf` to suit your situation.

Then tell squidGuard to compile its database:

```
# /usr/bin/squidGuard -C all
```

Sometimes you also need to add a no-block file to allow access to useful sites

Reload Squid

You need to tell Squid to reload its configuration before you can use squidGuard:

- If you install Squid from RPM (SuSE):

```
# rcsquid reload
```

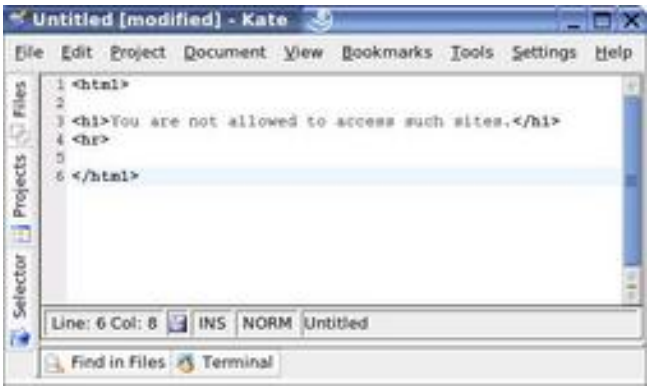
- If you install Squid from RPM (RedHat):

```
# /etc/rc.d/init.d/squid reload
```

- If you install Squid from tarball you can use the following command:

```
# /usr/sbin/squid -k reconfigure
```

Forbidden access page



SquidGuard in action

In this section, I will show a simple and useful squidGuard configuration to block porn sites.

squidguard.conf:

```
logdir /var/log/squidGuard
dbhome /var/lib/squidGuard/db
dest blacklist {
  domainlist blacklist/domains
  urllist  blacklist/urls
  redirect http://localhost
}
acl {
  default {
pass !blacklist all
redirect 302:http://localhost
  }
}
```

blacklist/domains:

```
069palace.com
0ver18.com
1-800-4hotsex.com
1-800-4sex.com
1-8004hotsex.com
1-xxx-live-sex-teen-pussy.nu
10000celebs.com
1000pictures.sinfulxxx.com
1000puresex.com
1001freepics.com
...
playboy.com
```

blacklist/urls:

```
00blow.com/analzone
00blow.com/menonly
0815.org/user/lolitainc
1-800-pussy.com/suzi
1-8004lovers.com/lessie2
1-and-only.com/members-only
100amateurs.com/bondage/images/domination
100amateurs.com/images/toys
100sluts.com/girlie
100teensluts.com/images/xpics
...
```

I redirect the blacklist to an HTML file.

squid.conf:

```
...
redirect_program /usr/sbin/squidGuard -c
  /etc/squidguard.conf
...
acl our_networks src 192.168.0.0/24 127.0.0.1
http_access allow our_networks
```

When I try to access the Playboy web site at www.playboy.com, for example, I get an error message.

Closing notes

To implement web site blocking you can use Squid on its own or with squidGuard. Each has advantages and disadvantages.

Squid's built-in mechanism's advantages:

- It's very easy to install and configure
- It's very fast, because it doesn't need external programs
- It's easy to debug. When the blocking mechanism isn't working correctly, you know which program is causing the problem

Unfortunately, it also comes with disadvantages:

- It's not flexible. It's very hard to configure forbidden domains and URLs categorization.
- Maintainability. When you have several ACLs for blocking sites, it can be a nightmare to edit or modify them to suit your changing needs.

SquidGuard has the following advantages:

- It's very flexible. You can categorize domains and URLs - as many as you like.
- It's maintainable. You can define many ACLs to suit your needs without too much trouble.

But it also comes with some disadvantages:

Error message when accessing Playboy web site

- It's not easy to install and configure
- It's not very fast because it's called from Squid
- It's harder to debug when there is a problem. In such a case, you have to check the Squid and squidGuard configuration.

You may encounter the following problems during your installation and configuration phase.

Db package not installed

- Install the appropriate version required by squidGuard

Squid is not blocking

- Check the logs (`access.log` and `error.log`)
- Check your `squid.conf` especially the acl rules
- Reload squid

SquidGuard is not blocking

- Check the logs (`squidGuard.log`)
- Check your `squidguard.conf`. If there is an error in this configuration file, squidGuard will not block anything.
- Check the permission for blacklists database. Squid must be able to read the files in the blacklists directory.
- Recompile squidGuard database with `squidGuard -C all`

Bibliography

Baltzersen, Pål and Lars Erik Håland, Configuring squidGuard, `/usr/share/doc/packages/squidGuard/doc/configuration.html`, 2002

SquidGuard Blacklist (`http://www.squidguard.org/blacklist/`)

Squid Configuration Guide (`http://squid.visolve.com/squid/sqguide.htm`)

Squid Documentation Project (`http://squid-docs.sourceforge.net/`)

Step By Step Install Guide for squidGuard 1.2.0 and BerkeleyDB 3.29 (`http://www.maynidea.com/squidguard/step-by-step.html`)

SquidGuard is free, very flexible, very fast, easy to install and portable

Copyright information

(The following license is effective immediately)

© 2005 by Tedi Heriyanto

This article is made available under the "Attribution-NonCommercial-NoDerivs" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-nc-nd/2.0/.

About the author

During the day, Tedi works as a system engineer and system analyst. He is also a contributing editor for several computer magazines in Indonesia. At night, he works as a computer programmer and security enthusiast. In his previous life, Tedi worked as a software development engineer and as a Linux training instructor.

Who's behind that web site?

SSL, certificates, and detecting phishers

John Locke

Let's talk about phishing. *Phishing* is just like fishing, only your identity is the fish and the bait is an email that looks like it came from your bank, or eBay, or Paypal, or any other legitimate place. The goal is to get you to follow a link to a site owned by the phisher, and trick you into divulging some private information, such as your bank account number, pin, passwords, or social security number.

Some phishing emails look completely legitimate, using logos, links, and text from the real business. Many try to warn you about fraud being committed with your account – the truth is, the senders of the email are the ones trying to commit fraud with your account, if they can trick you into divulging it. These types of emails are almost always fake. When you follow the link in such an email, you'll usually get taken to a web site that looks exactly like the real web site. But it's not.

How do you tell the difference between a real and a fake web site?

Stop! Wait! Before clicking any links, go update your web browser. In the past few months, many of the vulnerabilities in both Internet Explorer and Mozilla Firefox have been related to making it harder to tell what web site you're really visiting.

A malicious web site could use some carefully crafted web addresses that could lead you to believe you're really at a legitimate site. What's worse is that there have been some vulnerabilities in Internet Explorer that could allow these malicious sites to surreptitiously download programs to your computer. This is one of the prime ways you get infected with "spyware".

Figure 1. When Firefox has an update available, it shows a red arrow icon in the upper right corner

For Internet Explorer, go to Windows Update by following the link on your start menu. For Mozilla Firefox, look for a little red triangle in the upper right corner of your window, as highlighted in Figure 1.

> One of the biggest tricks of the phishermen is to add more symbols after a normal looking domain name, which makes your browser go to a completely different place than you're expecting

Okay, now that your browser is up-to-date, you're a bit more protected online. There are two basic things to pay attention to, when you're deciding whether to trust a web site:

1. The web address, or URL. URL stands for Uniform Resource Locater, but it basically just means the web address of the page you're viewing.

2. The security certificate. Also called the SSL or TLS certificate. We'll get into this shortly.

By applying what you know about web addresses and security certificates, you're better armed to detect a fraudulent site. So what do you need to look for?

Anatomy of a web address

Bear with me. We're about to get slightly technical here – but this is basic information you need to know to web surf safely. By now you've probably seen thousands of web addresses. There are some very strict rules for web addresses – how they are put together, what each part does, etc. For this discussion, we need to look at two of these parts: the protocol, and the domain name.

The protocol

The protocol is the very first part of the web address. It's the *http://* or *https://* part. You generally don't need to type it in when you go to a web site, because most browsers will add it for you. These are two different protocols used for web pages and several other types of data. A protocol, in this context, is a description of how to introduce yourself properly, the browser equivalent of knowing what words to say to introduce yourself, and ask for a book from a librarian or a fried banana from a Thai street vendor. In the web world, the dominant protocol is *Hyper-Text Transfer Protocol*, or HTTP. The Hyper-Text is your web page, the transfer protocol tells your browser how to get it. HTTPS is *Hyper-Text Transfer Protocol over SSL*.

SSL stands for Secure Sockets Layer, and lately it's been renamed to Transport Layer Security, or TLS. SSL/TLS is an entire framework that provides two things: encryption and authentication. Encryption hides your data between the web site and your browser, preventing anyone from intercepting it along the way. Authentication lets you verify that the server you're visiting is really the server it says it is, and not some other server taking its place. I'll get more into this in a moment.

So the first thing to check in the web address is, are you visiting a page that is protected by SSL/TLS? And you can tell this by looking at the protocol in the address bar of your browser – is it https, or http?

The domain name

The other part of the web address to look at is the domain name. The domain name is everything after the https:// part up to the next forward slash ("/"). And that means everything. One of the biggest tricks of the phishermen is to add more symbols after a normal looking domain name, that makes your browser go to a completely different place than you're expecting.

A domain name is broken up into domain "parts." The further right you go in the domain name, the more significant the part is. For example, looking at the domain name *www.freelock.com*, the most significant part is *.com*. It's significant for technical reasons, not just for the phrase "dot-com bomb".(should this be "dot-com boom")

Ready for another acronym? **.com** is called the *Top-Level Domain*, or TLD, of this domain name. Each country has its own two-letter TLD, such as .us, .uk, .au, .tv, and then there are the three-letter TLDs we're all familiar with: .com, .net, .org, .edu, .gov, .biz and a few other less common ones. These top-level domains are controlled by a designated registry, and copied into what are called the *Root Domain Name Servers*. There are 14 of them, scattered around the world.

Moving to the left in the domain name, the actual domain we're looking at is *freelock.com*. When your browser asks for www.freelock.com, it first goes to the Root Domain Name Servers to ask for where to find the directory for the domain freelock.com. The Root Domain Name Servers tell your browser to go to the IP address 69.55.225.251, which happens to be my Domain Name Server. Your browser then asks my name server where to find **www**, which could be an actual computer, or it could be another domain part.

If you're paying attention here, you might ask: what's so special about the "www" part? The answer is, absolutely nothing. It's just another domain name part, and usually takes you to exactly the same place as you would get if you left it out. A very small percentage of sites expect you to type www as the first part of the domain name; the vast majority take you to the same place without it. Hint for the lazy: don't bother typing www.

Bad domain names

So how do the phishers trick you? By putting something between the .com part and the first slash. Here are some

bogus web addresses I see in my quarantine, all targeting Paypal, a popular e-commerce site:

- *http://www.paypal.com-cgi-bin.biz/ppverify.php?cmd=_login-run&mail=&?motd=account_verify* – this link appears in an email apparently from Paypal. See the first slash after the http:// part? The TLD is .biz, and the primary domain part is com-cgi-bin.biz. Remember, anybody can get a domain name, and if you think about it, why would Paypal have a domain name called com-cgi-bin.biz? It certainly doesn't inspire trust.
- *http://www.paypalllaa.biz/ppverify.php?cmd=_login-run&mail=&?motd=account_verify* – Another phisher, registering a domain name like Paypal, hoping to trick you into visiting.
- *http://211.220.195.70/paypal/login.html* – Never, ever, ever, trust an IP address in a financial mailing. Four numbers instead of a domain name takes you to an otherwise anonymous machine on the internet. This is okay for your web developer, to show you a development version of your web site, but if anybody else gives you an address like this, you'd better have a good reason to trust them.

You can see these web addresses in the status bar (the very bottom strip of a program window) in most email readers. I say most, because there are a couple very popular email programs by a certain vendor that until recently thought it would be too confusing to show you the real link in the status bar. In these programs, which will remain nameless, it's very difficult to tell where a link really points. Yet another reason to switch to an open source email program, like Mozilla Thunderbird or Evolution.

But even if the address looks completely legitimate, you still have to be aware – there have been flaws in pretty much every mail reader that make it possible to hide the real destination of a link. The phishers use a couple of technical tricks that involve either embedding backspaces into the address, so that the real address gets overwritten by a fake one, or by using international characters that look just like the plain English text characters.

Worse, in March 2005, there were a series of attacks on the internet that were called "DNS Cache Poisoning." What happened was that many vulnerable name servers in use all over the world were fed bogus information about where to find the Root Domain Name Servers. Essentially, these name servers were hijacked. Imagine calling 411 to get a phone number, but instead of getting an operator at a legitimate phone center, you got a fake one that gave you the wrong phone number, on purpose, for the person you were trying to reach. By doing this, the bad guys could send you anywhere they wanted, and pull all sorts of tricks to make you believe you were really sent to your bank's web site.

This is where SSL authentication comes in.

Figure 2. Firefox warns you if there's a problem authenticating the SSL certificate

SSL/TLS authentication

As I said before, SSL/TLS does two things: encrypts traffic, and authenticates the server at the other end. The encryption part is simple, from a user point of view – if you see the lock icon in the bottom right corner of the window, the connection is encrypted. But without authentication, you could just be talking very privately with the garage-based scam artist posing as your bank.

That's why authentication is important. Authentication provides some assurance that you're at the real Paypal.com, and not a scam site. It works by checking something called a certificate that the web server presents to your browser, before sending any data. Your browser does a number of checks on the certificate, and if it appears to be valid, and matches the domain name in the address bar, it shows the lock icon in the status bar and gets the page. If it detects anything wrong, it gives you an authentication warning.

You've probably seen authentication warnings in your browser. Figure 2 shows one. There are several things your browser checks to determine if the certificate is legitimate:

1. Does the domain name in the certificate match the domain name in the address bar?

2. Has the certificate expired?
3. Is the certificate signed by a trusted authority?
4. Is the signing authority certificate valid, and current?

I'll get to certificate authorities shortly, but first, I'm going to point out what an SSL certificate DOES NOT DO:

- *A valid certificate does not guarantee that a site is legitimate!!!*

Just like a domain name, anybody can get a certificate. It costs a little more – between $35 and $800 a year, depending on the certificate authority – but it's easy to do.

A valid certificate provides a guarantee that somebody you trust (the certificate authority) has verified that the web site you're visiting really is the one in your browser's address bar.

SSL warnings would alert you if your DNS server has been hijacked, and you're visiting a fake site. It will also tell you if a web master is too cheap to buy a certificate from a trusted authority, or if they've been lazy renewing their certificates.

What's a certificate?

Public Key Encryption made secure electronic communications possible. Public Key Encryption involves two *keys*, or codes used to encrypt or decrypt data. One key is public, shared with the world at large. The other is secret, only stored on your computer. In public key encryption, whatever you encrypt with the public key can only be decrypted by the secret key – you cannot even decrypt it with the public key you used to encrypt it.

This may sound counter-intuitive, but think about the difference between multiplication and division – it's much easier to multiply two large numbers than to divide one from the other. This is the underlying principle that makes public key encryption possible.

In the other direction, you can use a secret key to *digitally sign* a chunk of data, and verify the signature with the public key. This turns out to be a very useful thing to use to guarantee the authenticity of a message.

A *certificate* is a public key, combined with information about the owner of the corresponding secret key, digitally signed by somebody else – the *Certificate Authority*. So when somebody wants to run an encrypted web server, here's what they do:

- Create a brand new pair of keys, public and secret.
- Send the public key, along with name, address, site name, and other details to a certificate authority, such as Verisign, GeoTrust, Thawte, or their local webmaster as a certificate signing request.
- The certificate authority uses their secret key to sign the certificate signing request, and the result is a signed certificate.
- The site owner installs the secret key and the signed certificate on the server.

Now, when you visit the site, here's what happens:

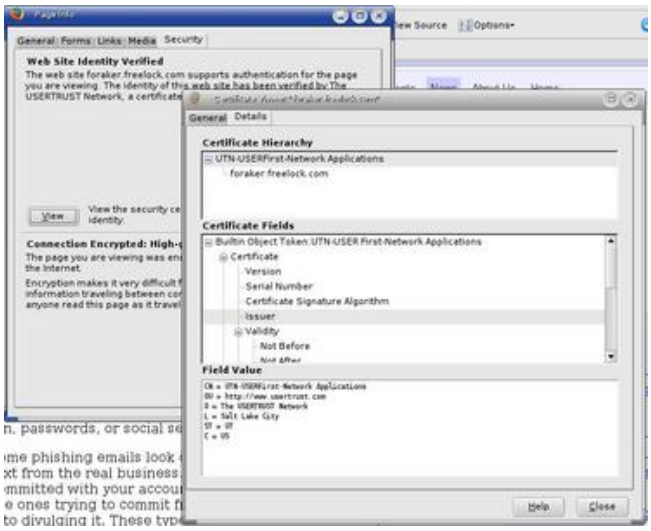
- Your browser connects to the server, and asks for its certificate.
- Your browser verifies the signature of the certificate authority. If it doesn't recognize the certificate authority, or the details in the certificate do not match the web address or have expired, or anything else weird, it pops up a warning.
- If the certificate is properly verified, or if you tell your browser to go ahead even though it couldn't verify the signature, your browser creates a new, random symmetrical key, encrypts it with the public key in the certificate, and sends it back to the server.
- From then on, both the server and your web browser have a big, shared key they use to encrypt all data going back and forth. Your browser will show the lock icon, so that you can easily see that your surfing is protected.

Who do you trust?

Trusted Authority. Certificate Authority. Signing Authority. All these are the same thing–a small set of organizations your browser has been preconfigured to trust. Their certificates are built into your browser or operating system. They're there because the people who created your web browser worked out a deal with a bunch of different organizations to have their certificates pre-installed. These are the companies like Verisign, Geotrust, Thawte, and others.

Anybody can run their own certificate authority, but if the signing certificate isn't built into browsers visiting sites signed with it, you'll get warned when you visit the site.

You can view the details of an SSL certificate, and see the "chain of trust" of signing authorities



Checking a certificate can help you judge whether the site you're visiting is authentic. To check a certificate in Firefox:

1. Double-click the lock icon in the bottom right corner of the browser window. This will open up the page info box, with the Security tab in front.
2. Click the *View* button to view the certificate.
3. On the *General* tab, you can see details about the owner of the certificate, the certificate authority who issued it, and validity dates. Check the very first item, the Common Name (CN). This is what your browser compares to the domain name in the URL. While the URL might be spoofed due to a browser vulnerability, chances are it will appear correctly here. Are there any strange characters in the Common Name? If you see any "^", or "%" symbols, watch out. Does the name actually represent the site you're trying to visit?
4. On the *Details* tab, you can see the whole *Chain of trust* in the Certificate Hierarchy pane. The first item in this list is a certificate authority built into your browser, who you trust to verify web sites.

Each of the items in the Certificate Hierarchy pane is a certificate. The first one is the one you trust. Sometimes there are up to four certificates in the chain. Each certificate has signed the next, vouching for it, telling you that it's legitimate. You can click on any of them to get more information about the particular certificate. The last one in the list is the one belonging to the server you're visiting.

How reliable is this system?

A phisherman can hijack your connection, and send you to a completely bogus web site. But at this point, they cannot forge an SSL certificate.

One of the principles of security is that no security is perfect – you just have different levels of protection. The entire system of public keys, certificates, and certificate authorities is collectively called a *Public Key Infrastructure*. Given that the encryption and authentication details are effective, there are still a few possible ways to defeat the system:

- Attackers could break into a server and steal the secret key. By installing it on a fake server, and using a different hijacking technique, they could possibly spoof the real thing. There's a mechanism called a *Certificate Revocation List* that allows certificate authorities to revoke a certificate, but this isn't fully working. So far, there haven't been many documented cases of this.
- Attackers could trick you into installing the certificate for a malicious certificate authority, and then your browser would trust whatever they told you to trust. Spyware could do this. This is perhaps the easiest way to break the system. Run anti-spyware software and keep your machine up-to-date to prevent this.
- Attackers could break into a real, trusted certificate authority, and sign a bunch of fake certificates. This would be like breaking into Fort Knox.
- Somebody might figure out how to break the encryption system itself, discovering a new mathematical technique. If this happens, I guarantee you'll hear about it all over the news – it would make the entire e-commerce system broken, and would affect everybody.

Seven steps to safer surfing

Based on the risks just listed, here is a summary of what you can do to safely conduct e-commerce on the internet:

1. Keep your system up-to-date with the latest operating system and web browser updates.

2. If you're on Windows, use anti-virus and anti-spyware software to keep your system clean.
3. Look for the lock icon in your browser anytime you're doing anything financial, or passing any kind of sensitive information.
4. If your browser alerts you to an authentication problem, pay close attention, and be extra cautious before doing anything sensitive.
5. Check the URL in the address bar, before typing anything sensitive – are you really where you think you are?
6. Compare the URL in the address bar with the Common Name in the SSL certificate, to be really sure of a site.
7. Listen for news of any break-ins to the sites you visit, or significant changes to e-commerce security in general.

Following these steps will keep you much safer online. But nothing is a substitute for having a questioning, critical mind. Don't trust anything you receive in email – verify it with the real source. Don't follow links blindly. Look in the address bar. The internet is a dangerous place, but armed with a bit of knowledge, you can keep your sensitive information safe. Now, if only the companies you do business with could do the same...

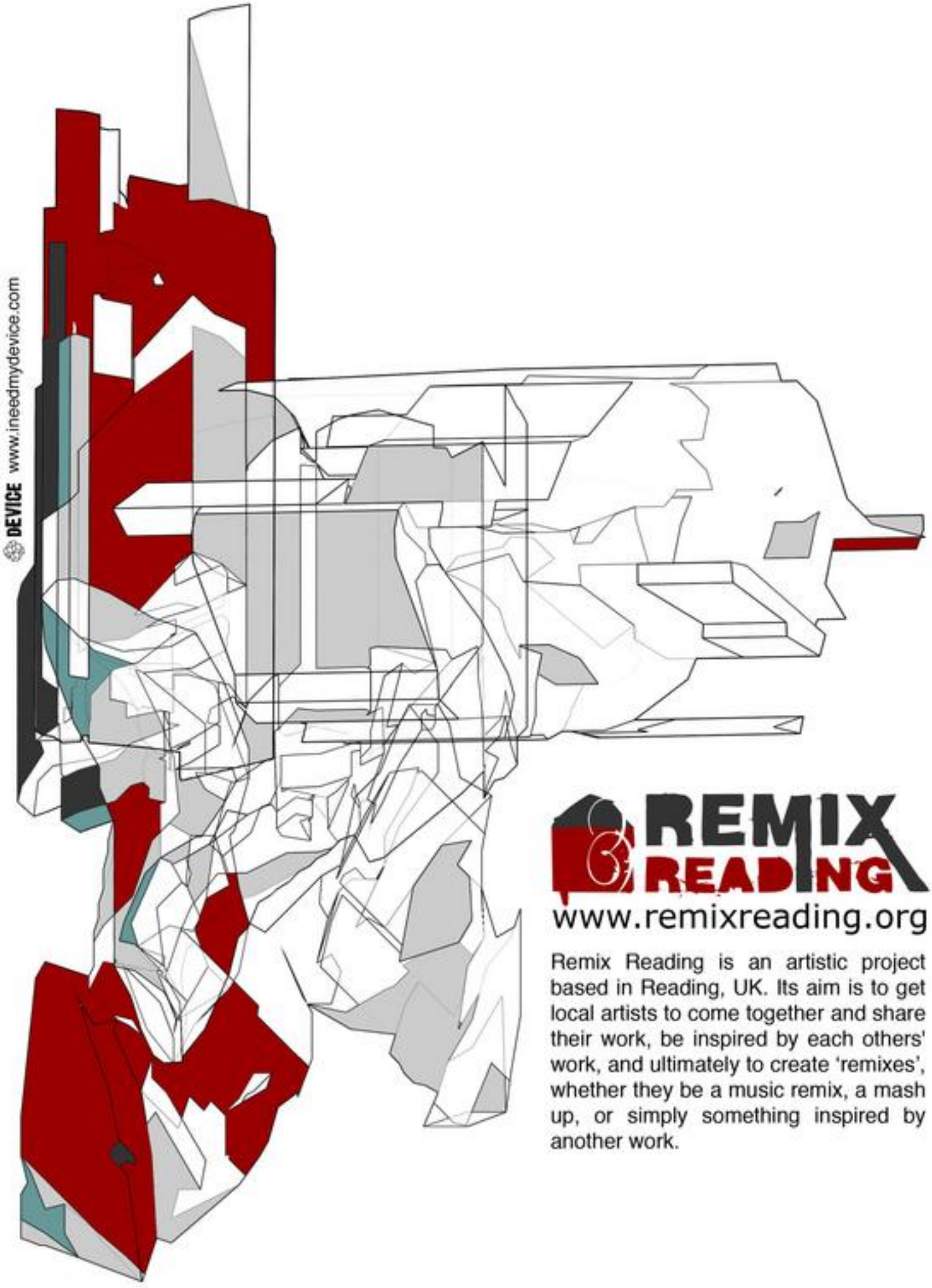
Copyright information

© 2005 by John Locke
This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

John Locke is the author of the book *Open Source Solutions for Small Business Problems.* He provides technology strategy and free software implementations for small and growing businesses in the Pacific Northwest through his business, Freelock Computing (http://freelock.com).

A PLANT NEEDS WATER TO GROW
FREE SOFTWARE MAGAZINE
NEEDS SUBSCRIBERS TO PROSPER!
Free Software
MAGAZINE
BY SUBSCRIBING YOU WILL BE SUPPORTING A MAGAZINE WHICH BELIEVES IN FREE SOFTWARE. ALL OUR ARTICLES ARE RELEASED UNDER THE GNU FREE DOCUMENTATION LICENSE, ENHANCING EXISTING INFORMATION ON FREE SOFTWARE.
SUBSCRIBE NOW!
HTTP://WWW.FREESOFTWAREMAGAZINE.COM/SUBSCRIBE



REMIX READING

www.remixreading.org

Remix Reading is an artistic project based in Reading, UK. Its aim is to get local artists to come together and share their work, be inspired by each others' work, and ultimately to create 'remixes', whether they be a music remix, a mash up, or simply something inspired by another work.

The future of computing: is free software ready?

Be it on-demand or in-house, free software works for all

Kaustubh Ghosh

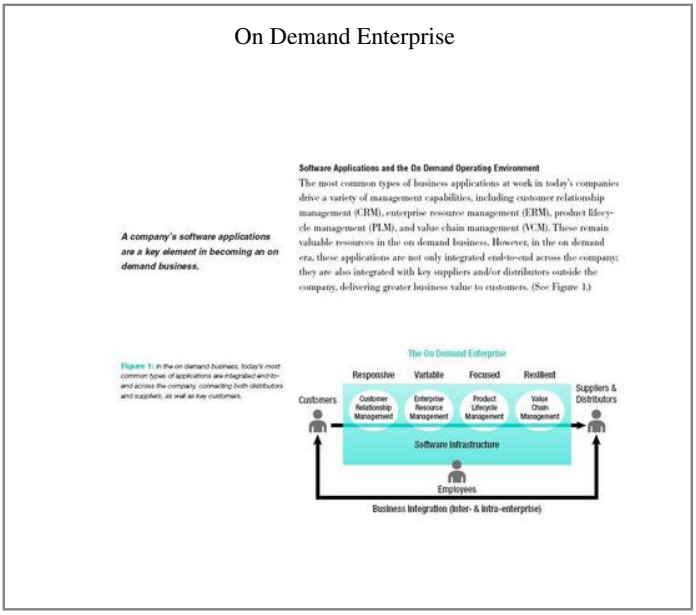
The future is the state of things yet to come. One can only expect what may happen and never know what will happen. The future can only be predicted based on past experience. The predictions differ based on the forecaster and his experience, in-depth understanding and knowledge. The technological future is the technology of the future, the destiny of the technology of today.

> Microsoft's idea of user empowerment suits a wide range of users, including the ones left out by IBM

The predictions of tomorrow's technology are as diverse and conflicting as any ranging from the on-demand business model promulgated by IBM to the user empowerment as envisioned by Microsoft. While the top players in the field continue composing and refining their visions and trying to make them reality, the debate rages on.

IBM's vision

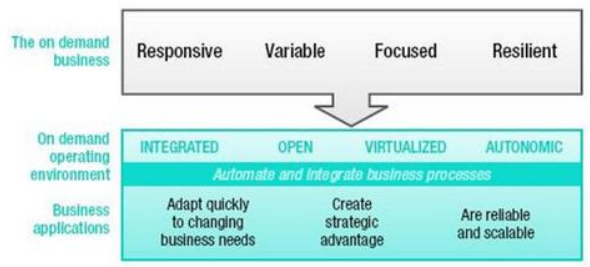
IBM predicts that users of technology in industries ranging from the small scale to mass production based factories, like television manufacturers, will lose interest in implementing services on their own, what Microsoft terms in-house. The basic idea is the user has to pay full price even for the software he does not need or rarely uses. So


On Demand Enterprise
A company's software applications are a key element in becoming an on demand business.
Software Applications and the On Demand Operating Environment
The most common types of business applications at work in today's companies drive a variety of management capabilities, including customer relationship management (CRM), enterprise resource management (ERM), product lifecycle management (PLM), and value chain management (VCM). These remain valuable resources in the on demand business. However, in the on demand era, these applications are not only integrated end-to-end across the company; they are also integrated with key suppliers and/or distributors outside the company, delivering greater business value to customers. (See Figure 1.)
The On Demand Enterprise
Responsive
Variable
Focused
Resilient
Customers
Customer Relationship Management
Enterprise Resource Management
Product Lifecycle Management
Value Chain Management
Suppliers & Distributors
Software Infrastructure
Employees
Business Integration (Inter- & Intra-enterprise)

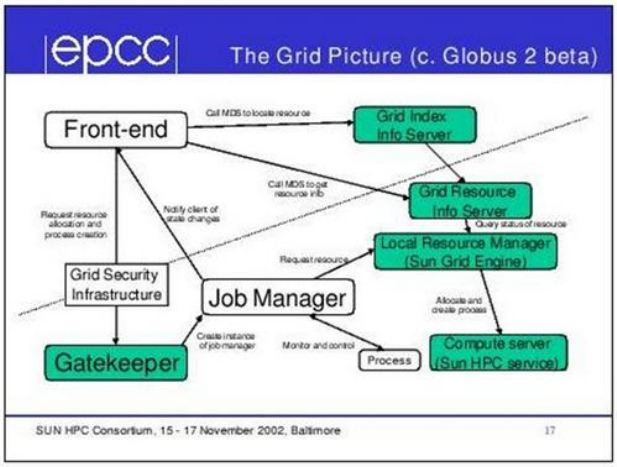
the better option for the user is to pay for what he uses, as long as he uses. IBM believes they will buy computing services and support from technology vendors. Users can then use only what they need and pay for only what they use – on demand. The idea they explain with great detail here (http://www.devx.com/assets/ibm/5481.pdf). IBM is already offering the on-demand kit for download (http://www-5.ibm.com/e-business/za/kit/ondemandkit.html).

"A year ago, it was an assertion. We've moved well beyond assertion into adoption and reality" said IBM CEO Palmisano, of the on-demand effort during a keynote ad-

Interrelationship



Sun Grid



dress at the IBM Business Leadership Forum in San Francisco. IBM itself expects to cut $7 billion in expenses from 2003 to 2004 by employing the idea, Palmisano said.

Big Blue expected one outsourcing area of its on-demand work, in which IBM takes over components of a company's business, to be a $150 billion market back in 2004 and to grow 12 percent to 14 percent each year for the next five years. UPS, Procter & Gamble, Hewitt Associates, FedEx and Lincoln Financial Group are among the IBM customers for this "business transformation outsourcing" area.

Microsoft's vision

In contrast to IBM, Microsoft sees the individual user as the design point. Microsoft believes that further empowerment of the user as an individual is the key to the next generation of computing. The company equates the on-demand model with the time-sharing model and contends that outsourcing computing services minimizes the individual persons control over the computing resources. Instead the future will be driven by individual users on personal computers using standards based tools with Microsoft's enhancements for everything from simple tasks to intricate applications.

The concept is ideally disposed for Microsoft's "per box" sales strategy and works perfectly for its marketing strategy to maintain domination in the PC world. The open standards in the software industry provide the essential base for the development of Microsoft products but it is the proprietary closed source code that shapes the main structure. The development of powerful tools, embracing open standards and enriching and enhancing them with proprietary closed extensions, provides the core of Microsoft's vision of the future of computing.

What others are thinking

The majority of the companies are thinking in line with IBM. Even Sun, which once shunned services in favor of selling big, expensive boxes, is the latest to jump on the pay-per-use utility computing bandwagon with a new grid offering that promises to bring computing power to users on demand.

It's not, however, the first move Sun has made in the utility computing arena. Sun's N1 strategy, unveiled more than two years ago, is aimed at helping customers pool resources so they can grow and shrink according to business demands. However, Sun Grid compute and storage utility offerings represent a shift for Sun because they will be delivered on a hosted basis as is described by sun on their web site (http://www.sun.com/third-party/global/bea/pr_030104_01.html). Sun's strategy is also dealt with in detail here (http://www.pcw.co.uk/news/1158266).

HP is also thinking with the trend with its Adaptive Enterprise Strategy. It is a road map that defines how the company will integrate hardware, software and services to help customers respond quickly to changing resource needs and thus help their organizations run more efficiently.

Company executives describe three stages that IT managers must tackle and complete before becoming truly adaptive, by HP's definition. The first stage, as per Nora Denzel, senior vice president of HP's Software Global Business Unit, is to assess their networks and retool, re-architect and re-engineer their infrastructures to support automation and service management across networks, servers, storage and applications. The second stage IT managers must overcome to become adaptive is business efficiency, in which network elements are managed as business services, and the third, dubbed business agility, is when the software and hardware infrastructure dynamically adapts to meet the changing needs of the business. The strategy is given in detail on HP's site (http://www.hp.com).

An analysis of the views

On the superficial level, all of the proponents seem to be correct for their own respective targeted markets. IBM targets the not so tech savvy organizations wary of implementing technological services on their own as does Sun. Their embracing of on-demand services provided by an organization is beyond doubt. But organizations that need to control their infrastructure for competitive edge or high-end, mission-critical reasons, like satellite control, space missions or military security, is guaranteed to opt out of this model. They are not in the technological future as propounded by IBM and its compatriots.

On the other hand, Microsoft's idea of user empowerment suits a wide range of users, including the ones left out by IBM. With proper GUI-based tools and simplifying interfaces Microsoft has shown that even techno phobic organizations can implement their required services with ease. But lying under the appearance of user empowerment by direct sell and update strategy is the trap door of high cost and complete dependency upon the developer company. Besides, with a long history of security vulnerabilities and bad reputation the company's software is always under threat from virus coders. Once support for a product ends there is no other way but to upgrade. And if you can't upgrade, with the unleashing of a new virus, you can only pray for a highly skilled and capable programmer familiar with the architecture to develop a patch and provide it free or for a small fee.

Free software suits all

It is at this crucial juncture that free software comes to the rescue. Applications built for the on-demand computing platform, embraced by smaller techno-cretinous organizations, can be easily shifted to in-house, in-control environments to suit the high-tech organizations as long as free software is used in both of these extremes. Under this condition the contrary is also true: applications built for the in-house model can be easily shifted to the on-demand environment. It is the universality of free software, which really empowers the user, allowing access to the core of technology and services on offer, thereby bestowing the ultimate power possible in market - the ability to choose and change suppliers depending upon user need.

The strategy also ignites the creative passion of the programmers, developing software unfettered, free of the enforcements of narrow ownership. Even Linus Torvalds has no claim of proprietorship beyond his "Linus tree" and anyone can start their own tree. It is the all pervasiveness of free software, encompassing the entire range of on-demand services as well as in-house computing, that makes it the main contender for the techno-solution title belt and has earned it its place in the sun. Based on its merits we can rightfully proclaim that it *is* the technology of the future.

Copyright information

© 2005 by Kaustubh Ghosh

This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

Kaustubh was born and raised in Kolkata (former Calcutta - the city of joy), India. He is currently studying Computer Science & Engineering at Jadavpur University, Kolkata (fourth year). He is the moderator of Jadavpur University GNU/Linux Users Group and is associated with "Computer Jagat" magazine published at the University. He is also working on a project aimed at developing a Steganographic File System for Linux.

Free software 2.0

Commercial class software with free license flexibility

Nathaniel Palmer

Free software (and open source) license models have become the most influential force in business IT to date. The first part of this article presents a brief history of free software, combined with the findings from an analysis of the attitudes and expectations, across several hundred large and medium-sized businesses, relating to free software. The second part of this article presents Delphi Group's vision for the next wave of commercial free software, where demand is driven not by cost alone, but foremost by quality of service and increased agility.

Free software 1.0 - a brief history of free software

The first commercial computers came with "free" software, including the source code, which could be freely shared. It wasn't until the 1970s that independent commercial software became widely available. By this time, competitive forces had lead to increasingly closed-source architectures and restrictions on redistribution.

The decades that followed saw both explosive growth in software development and rapid declines in the cost of computing power.

Hardware moved rapidly toward commodity status, while software became increasingly proprietary and redistribution restrictions were enforced at an increasingly aggressive rate. This trend led to two sets of circumstances whose repercussions now hold the potential to redefine the software industry:

1. communities of programmers have ready access to hardware horsepower, but find the latest and greatest software tools out of reach;
2. commercial developers seeking to differentiate their applications developed increasingly closed and proprietary software.

By the late 1970s, the growing cost of software first inspired the early seeds of today's free software movement, including the GNU Project and the Free Software Foundation. For decades to follow, the free software movement grew within communities of hackers who viewed commercial software as a cultural anathema. Yet the innovations these communities produced were largely relegated to command line UNIX, with little impact on end user computing or commercial software sales.

In the 1990s however, the free software trend line hit an inflection point. As a result of maturing standards (such as HTML and XML), the open orientation of the internet, the evolution of Java and J2EE, and the success of initiatives such as *Apache, Linux*, and *MySQL*, the terms "open source" and "free software" have become part of the modern business vocabulary. Today, free software represents one of the greatest opportunities for both buyers and sellers of software, offering what is increasingly viewed as a viable exit strategy away from the trappings of the software oligarchy and the rising cost of proprietary licensing.

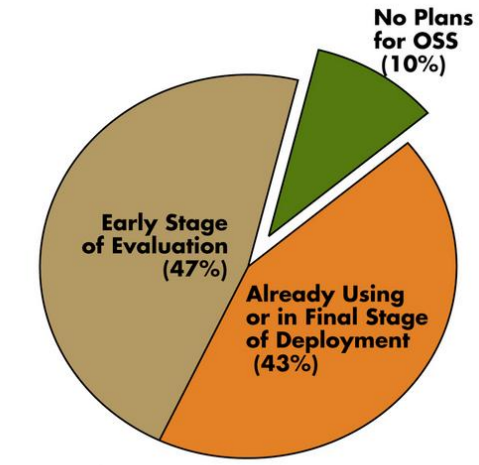
Shifting sands across the software landscape

In a Delphi Group survey of several hundred software consumers, nearly half of respondents agreed with the statement that free software represented *"an emerging area about to revolutionize the software industry,"* while 10% cited free software as *"already the best way to go for software development and procurement."* Interestingly, when the same questions were put to software developers, the responses proved even more favorable to free software.

In March of 2004, Delphi Group surveyed several hundred large and medium-sized businesses, as well as government agencies and large universities, regarding their use of and attitudes toward free software. The organizations examined included large consulting firms such as EDS, Accenture, and CSC; large medical services firms such as the Mayo Clinic and Carle Clinic; engineering and manufacturing firms such as Alcoa, CDM, and Elekta; firms in the energy sector such as SPL WorldGroup and Santos; United Stationers Supply Company, North America's largest business products wholesaler; pharmaceutical firms including Lattelekom and Novartis; publisher Penton Media; as well as non-profit firms such as Mercy Ships. Also surveyed were several universities and government agencies including the Federal Aviation Administration, the U.S. Congress, and the State of Nevada's Department of IT. The responses presented in this article reflect only those of users and internal application developers (e.g., software consumers). Responses from commercial software developers, resellers, consultants, and integrators have not been included in the analysis illustrated in this article.

The free software licensing model allows software to be freely shared, shifting competitive differentiation among software publishers from proprietary code to the quality of support and services. By shifting the point of competitive differentiation from proprietary code to openness and adherence to standards, free software holds the potential to radically alter the economic equation that has defined the software industry for the last 3 decades. There are few, if any, developments in commercial computing, which have evolved as quickly as the adoption of free software. This is driven by a combination of internet-delivered software (i.e., downloadable code), maturing standards, and of course, a price point that is either free or virtually free compared to closed-source equivalents.

The stage of deployment for free software



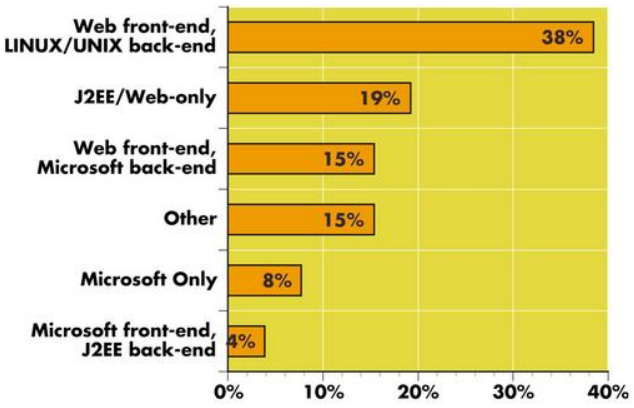
© Delphi Group 2004

The first wave of free software to impact commercial computing was at the server level, notably the explosive rollout of Linux since the late 1990s, and the less visible but equally pervasive adoption of the Apache Web server family. It wasn't, however, simply the low (or arguably nonexistent) price point that paved the way for these and subsequent free software initiates. It was the democratization of application development by J2EE and its constituent set of community-driven standards. For this reason, most commercial adoption of free software has been on the J2EE and/or Linux/UNIX platforms.

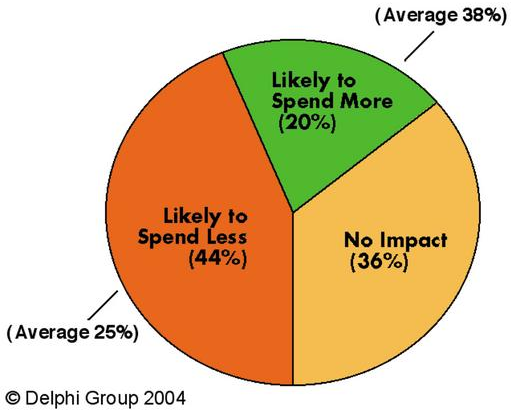
The most commonly deployed free software today includes the *Apache Jakarta* projects (Java-based server-side tools), the *Apache Tomcat* servlet engine, the *Eclipse* Java-based development environment, the JBoss J2EE application server, and tools such as the PHP scripting language and the *Samba* Windows-to-Linux integration software. The most widely deployed free software is Linux, which is today also the most widely installed UNIX variant and by far the one with the most rapidly growing market share.

The object-oriented, component-oriented, standards-based architecture of J2EE has hastened the development of multiple free software project initiatives, such as those under the *Apache Software Foundation (ASF).* The ASF community represents over 1,000 core developers and over 50 active projects. Organized as a nonprofit association, ASF is a meritocratic community governed by a core set of laws and principals (known as *Foundation Bylaws*). Projects are

The deployment environments where free software has or will be used



What impact will the availability of free software options have on 2004 IT spending and budgets?



vetted and approved through a peer-review process involving both users and developers, comparable to what would be found in the QA process of commercial software development.

Spending impact: economic consequences of free software

One of the fundamental arguments forwarded by free software detractors, is that without commercial incentives based on proprietary intellectual property, software innovation will cease and ultimately software consumers will suffer.

The economics behind ASF and other community-based free software initiatives are that of reciprocity. Specifically, the contributions to the development of free software will be rewarded with access to works provided by other developers.

The argument that free software removes economic incentive, however, is contradicted in part by the fact that 20% of the firms surveyed indicated they would spend more on IT as a result of available free software applications. Expectations for increased spending are likely the result of the lower entry point offered by free software, rather than the expectation that free software will be more expensive than proprietary alternatives.

Twice as many firms indicated they expect to spend less, which might be taken as fodder for free software detractors. This statistic likely ignores the fact that with a lower entry point, more projects will begin and ultimately these projects will lead to more software-based business activity. What is indisputable, however, is the fact that free software is already resulting in major changes on both sides of IT spending, with only 20% of firms citing "no impact" as a result of the availability of free software options and alternatives.

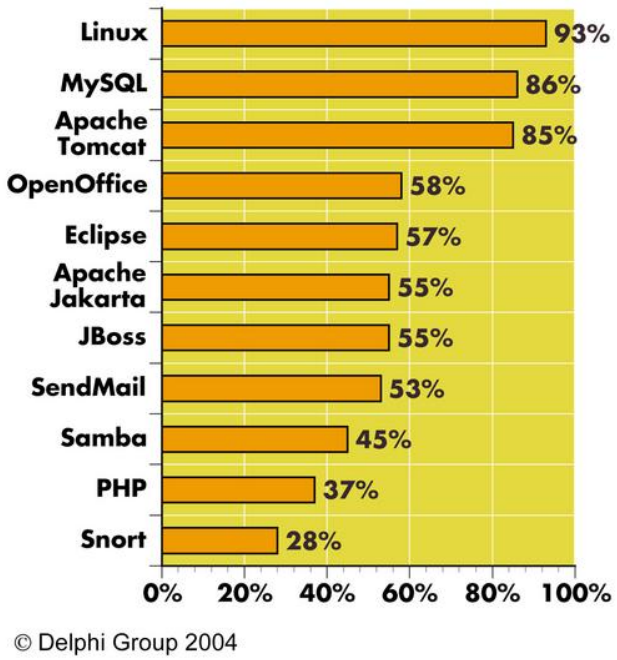
Free software 1.5 – free software; commercial services

While the first few decades of the free software movement saw little commercial traction, the first real commercial opportunities evolved through the offering of free software (what I will refer to as "free software 1.5") with billable services around maintenance, support and custom configurations. The commercial model behind this generation of free software is more about services than software. Software is the result of a community of developers, not a single commercial vendor.

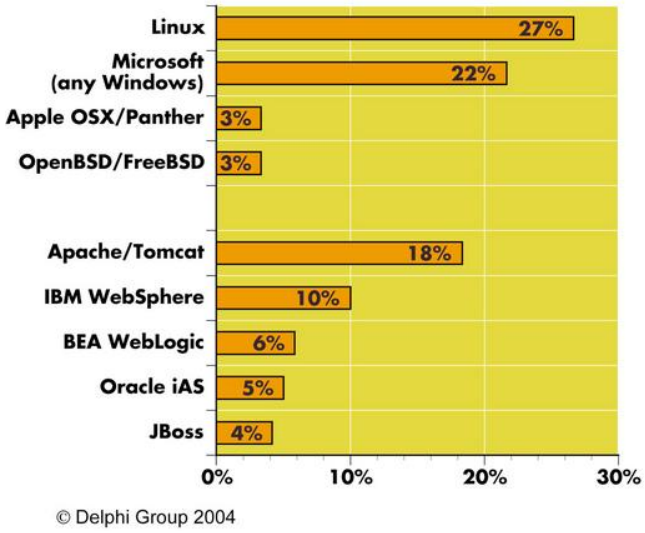
The first wave of commercial free software adoption has been defined largely by server-side tools and infrastructure elements, notably Linux and Java/J2EE-based projects. The business models for vendors in this generation of free software (such as JBoss and MySQL) are based on consulting services and support.

When differentiation and specialization come into play, it's not with regard to software development, but rather the services around a specific free software project, such as Linux. For example, commercial firms engaged in this generation of free software (such as *Redhat* and *Jboss*) provide free ac-

What free software initiatives do you use today or are likely to use tomorrow?



The anticipated platform for the next free software deployment



cess to free software and charge fees for services such as technical support, software customization, and application development.

The free software infrastructure "stack"

Adoption of the first generation of commercial free software has been defined less by packaged applications than by piecemeal components as part of a larger Web infrastructure. The continued development of both standards and prepackaged components has followed the growing demand for a complete infrastructure "stack" or set of foundational services and capabilities. Perhaps the first available free software stack was the Internet Application Platform (IAP) comprised of a core set of components commonly referred to by the initials "LAMP": *Linux, Apache, MySQL, PostgreSQL, PHP* and *Perl*.

LAMP offers a baseline of capabilities to rival proprietary alternatives, such as *SunONE, IBM WebSphere, BEA WebLogic*, or *Oracle iAS* (each of which leverages free software components, yet do not pass on to consumers the benefits of free software licensing). What it provides is a basic set of foundation services rather than addressing the more sophisticated requirements for today's organizations seeking to develop composite applications.

Enabling dynamic business applications requires moving "up the stack" with additional capabilities focused on a more robust presentation layer, as well as access provisioning, process execution and information management tools. Delphi defines the *free software infrastructure* stack as consisting of a core set of components loosely grouped into two sets of capabilities:

Value-Added Capabilities

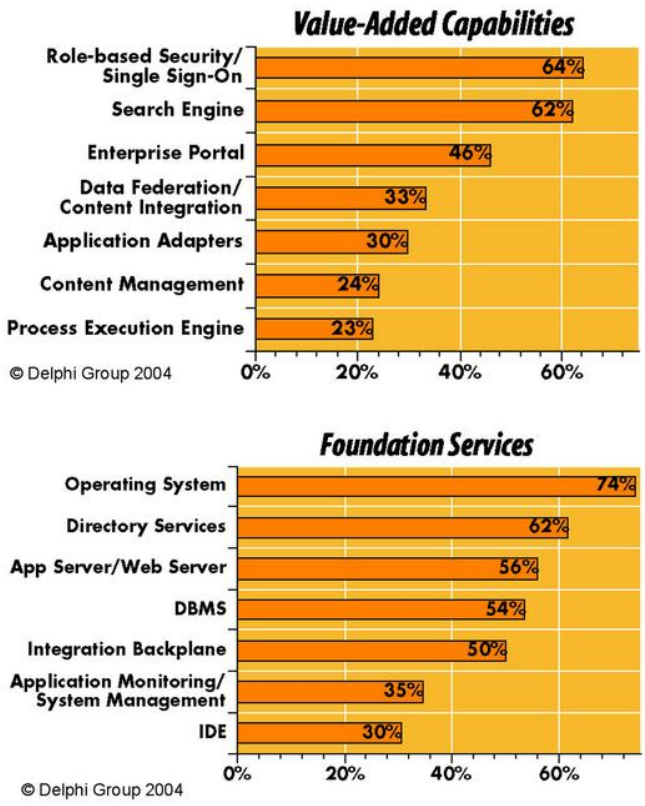
Application Adapters, Content Management, Content Integration and Data Federation, Enterprise Portal (presentation layer and UI), Process Execution Engine, Role-based Security/Single-Sign On, Search Engine

Foundation Services:

App Server/Web Server, Application Monitoring/System Management, Development Environment (IDE), DBMS, Directory Services, Integration Backplane, Operating System

Illustrated in the charts below, respondents were asked to rank the free software capabilities and components listed, based on what they deemed as necessary for a "complete" free software infrastructure stack. The foundation services at the bottom of the stack provide the basic building blocks and server-side resources for running, managing, and inte-

Components deemed either "absolutely necessary" or "significantly important" in the free software infrastructure capabilities "stack"



grating applications. These capabilities provide the "horsepower" for a free software infrastructure stack. For these capabilities, the commercial drivers are largely rooted in cost reductions – they're free to acquire and arguably cheaper to run and manage. These were largely developed and supported through the system of reciprocity described earlier in this article. For commercial providers of free software infrastructure stack foundation services, the business model is based on value-added services, ranging from support and customization to management and delivery of software updates.

As these capabilities further commoditize the market for server-side software (such as operating systems and application servers), the point of competitive differentiation, among both traditional software "stack" suppliers *(IBM, Sun, BEA Systems, et al.)* and free software infrastructure projects, continues to move up the stack into value-added capabilities.

Free software has reached the point where it provides an equivalent to every proprietary package on the market today. It is not surprising, however, that the evolution of enterprise free software packages have followed a different route than closed source. ERP packages, for example, represent a space where free software tools have been slow to evolve, particularly compared to the much faster evolution of free software development tools and infrastructure. This is no doubt partly cultural and partly economic, as the value proposition of ERP has long been rooted in a compromise between prepackaged functions and best-of-breed alternatives (i.e., having everything already in one package rather than spending the time required to separately source and integrate best-of-breed alternatives).

In contrast, the value of free software has been firstly cost and secondly the flexibility gained through transparent source code. For these reasons, free software has arguably represented the antithesis (and perhaps for some the antidote) of ERP. So it is not surprising ERP itself has been slow to emerge within the free software community. Yet, the upside to ERP packages, notably the opportunity for pre-integrated, commercially vetted software modules, is not entirely antithetical to free software. Rather, I believe, this will in fact represent the next wave of free software adoption – free software of a commercial quality delivered through a subscription-based business model.

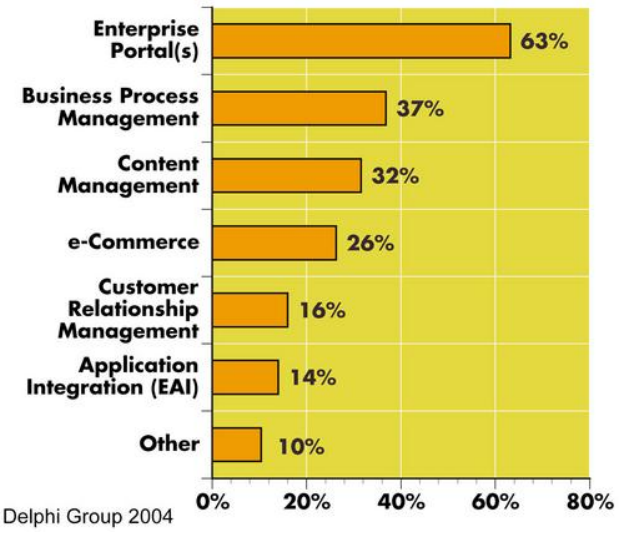
Shifting the target from "cheaper" to "better"

As indicated by the chart below, firms deploying free software are targeting the enterprise portals by an overwhelming margin (better than 2-to-1 over any other application area). "Managing business process" was cited half as frequently, but also twice as often with integration middleware (EAI).

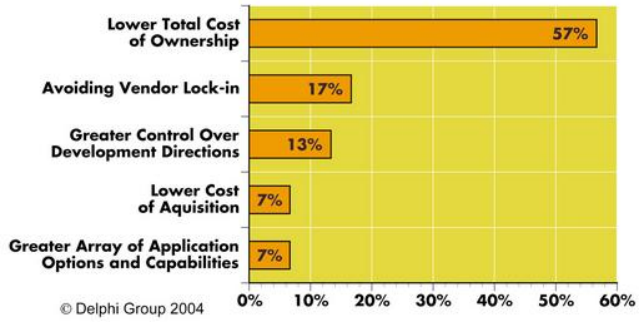
These data points help validate that firms increasingly see free software licensing and free software infrastructure stack investments as an environment for building, managing and maintaining collaborative web applications, rather than simply running server-side software (as is executed by the foundation services in the lower half of the free software infrastructure stack).

Firms embracing free software and infrastructure today are most often doing so in order to build collaborative, web-

The application areas where free software will be applied



The single greatest benefit motivating the adoption of free software



based applications. These firms are pursuing free software options primarily to realize a lower total cost of ownership.

For this reason, the manageability of free software has become a key consideration for commercial free software investments. Decisions are not made simply on the basis of what is cheaper to acquire, but more so on the long-term financial consequences and benefits of ownership. This point is validated by the fact that more respondents cited "*lower total cost of ownership*" by a ratio of nearly 10-to-1 over "*lower cost of acquisition*" when asked to identify the single greatest driver for free software investments.

Just as the bottom of the stack has benefited from standardized J2EE protocols such *as JAAS, JTS, JMS*, and *JNDI*, standards represent core drivers for value-added capabilities and enabling a lower cost of ownership for free software infrastructure. Recently, the coalescence of a set of both J2EE-derived and service-oriented standards have emerged which greatly enhance the ability to build and deploy both user-facing and distributed applications on the free software infrastructure stack.

These standards help to simplify the integration overhead required with combining individual free software projects, into a more unified framework or facilitate connectivity between dedicated software functions, such as process execution and security/access provisioning. Examples include *WSRP (Web Services for Remote Portlets),* which allows users to customize portal environments through point-and-click integration of standards-based portlets from any WSRP-compliant portal server.

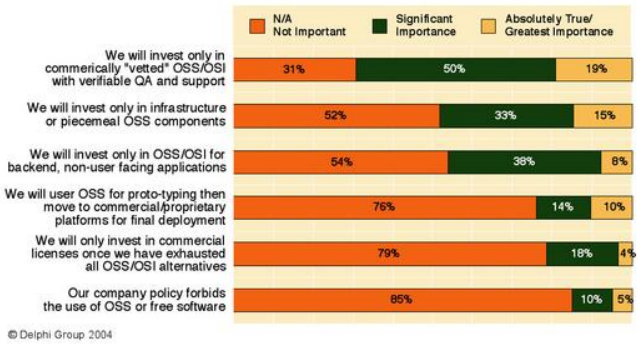
WSRP and distributed, standards-based, access control mechanisms such as *XACML (eXtensible Access Control Markup Language)* enables more administration functions to be delegated to users, while still maintaining centralized governance policies.

Leveraging these standards to delegate administration and empower users, allows for a decreased reliance on system administrators and other IT staff, without comprising content or data security. In many organizations, this should open the door for significant advantages in the reduction of software ownership costs. For these reasons, as well as increased portability of development efforts, standards will remain central to driving free software adoption. Neither standards alone or free software licenses, however, fully guarantee reduced ownership costs. Given that the greatest component of ownership costs comes from the cost of labor (both the personnel and services required for customization and support), free software that is free to obtain but labor-intensive to manage, presents no economic advantage over proprietary alternatives subject to more rigorous QA processes.

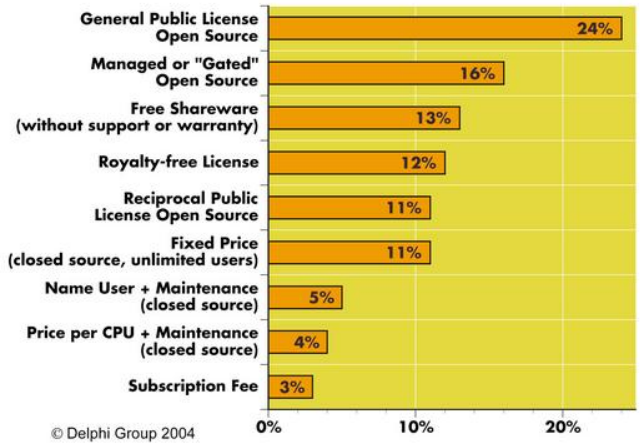
Free software 2.0 - commercial class quality, free software advantages

For free software to be readily adopted by a majority of commercial software consumers, offerings must be able to demonstrate a standard of quality consistent with proprietary, commercial alternatives. This arguably exists already

The ranking of attitudes towards free software adoption



The preferred licensing model for the next enterprise software investment



at the component or project level, but must also be demonstrable up the entire stack of free software infrastructure.

One of the early barriers to commercial free software adoption was organizational policies that forbid the use of free software. This issue has largely waned, while acknowledgment of the need for commercially vetted free software dominates adoption criteria. Commercial software buyers today seek free software solutions with demonstrable quality assurance processes, and confirmation that integration complexity otherwise involved with combining piecemeal software components has been successfully addressed.

An overwhelming majority of responses (70%) cited the need for the commercial vetting of free software offerings as the leading requirement for investment and adoption. This requirement is consistent with the 10-to-1 favoring of ownership costs over acquisition costs in the identification of perceived free software benefits.

The ability to provide commercial class quality within free software is a function of both the licensing model and delivery mechanism for bringing software to market and into the hands of commercial buyers.

Free software licensing models: myths and realities

For many, "open source" and "free software" are understood to be synonymous with "no cost". As a result it is often erroneously assumed that free software cannot be sold. However, free (and open source) software mustn't be confused with "shareware" and "royalty-free" licenses. "Shareware" software is normally freely distributed software with voluntary registration fees. Sometimes, you pay to get extra functionalities. This model does restrict the salability of derivative works, and the source code typically is not distributed and cannot be modified or embedded. A similar model is "royalty-free" licensing, which provides for a one-time right to integrate code with other applications (e.g., an embeddable software engine) and often does provide the right to resell derivative works, but typically does not provide access to improvements made to the original source code. In contrast with these two models, however, are free and open source software license models that allow free access and distribution to source code, while also supporting a spectrum of viable software business models.

Without limitation to specifically targeted free software investments, by a significant margin firms indicated a preference for "free software" and "open source software" licensing models for their next enterprise software purchases. Although the implied endorsement of free licensing may belie the more frequently stated requirement for stable, tested, and commercial class software, these findings underscore the growing momentum away from proprietary, named-user license models.

The defining characteristic for free software licensing is not the inability to charge a fee for software, but access to source code.

Each free software project family (e.g., ASF, GNU, BSD, etc.) has developed its own specific licensing policy, but all are based on a variation of one of the following:

- *Reciprocal Public License* - provides reciprocal rights to all changes, but does not allow for resale or licensing of derivative works;
- *General Public License (GPL)* - free to install and share, but does not allow for proprietary change or licensing of derivative works;
- *Lesser General Public License (LGPL)* - allows components of free software to be embedded in proprietary licensed applications;
- *Managed free software licensing* - combines support and maintenance benefits of licensed software, as well as ownership and licensing of derivative works.

The model which most closely follows the requirements for commercial software consumers today is: *managed free software*.

Free software 2.0 = "managed free software" licensing and delivery

Managed free software licensing offers a combination of support services and software updates to free software. Under this model a commercial vendor assumes the responsibility for testing and validating (either internally developed or community derived) software updates, delivered over network-based web services.

Early variants of this model include the *Red Hat Network*, which offers subscription-based services for delivering updates to the Red Hat Linux platform, and *Gluecode Software,* which offers a model of package applications and infrastructure.

Gluecode focuses on the top layer of the free software infrastructure stack, delivering value-added capabilities (on top core free software foundation services) including process management, security management, and an enterprise portal framework. These sets of capabilities are delivered through a pre-integrated set of software components, delivered as a locally-deployed server *(Gluecode Enterprise Server).* Gluecode has partnered with the Apache Software Foundation (ASF) to validate, extend and package a set of free software projects (notably those falling *under Cocoon, Jakarta*, Portals, web services and XML-specific project families) which address the top layer infrastructure requirements for the development of collaborative web applications. The *Gluecode Managed Free Software* service provides a delivery platform, which combines the Gluecode development methodology with automated software delivery.

Conclusion

Free software is already shifting the power curve from the software vendor back to commercial software consumers.

The recent research presented in this article illustrate that a majority of firms already view free software as a strategic lever to lower the cost of development and maintenance of enterprise software. This is based not simply on the availability of "cheaper" software, but the expectation of a lower total cost of ownership enabled by the combination of free software flexibility and commercial class capabilities.

Copyright information

© 2005 by Nathaniel Palmer

Verbatim copying and distribution of this entire article is permitted in any medium without royalty provided this notice is preserved.

About the author

Nathaniel Palmer is Delphi Group's Chief Analyst and Vice President. He is also the director of the Business of Technology practice, where for over a decade he has helped define the strategic positioning and market strategy for some of the industry's most visible leaders. Nathaniel shapes much of the Delphi Group's thought leadership, is the co-author of *The X-Economy* (Texere, May 2001) and has authored over 200 studies and published articles. His insights can be found in publications ranging from *Fortune* to *The New York Times*. He is also the Association of Information Management's first recipient of the Workflow Laureate and was recognized as Master of Information Technology.

On the "Creative Commons": a critique of the commons without commonalty

Is the Creative Commons missing something?

David M. Berry Giles Moss

On the face of it, the Creative Commons project appears to be a success. It has generated interest in the issue of intellectual property and the erosion of the "public domain", and it has contributed to re-thinking the role of the "commons" in the "information age". It has provided institutional, practical and legal support for individuals and groups wishing to experiment and communicate with culture more freely. A growing number of intellectual and artistic workers are now enrolling in the Creative Commons network and exercising the agency and freedom it has made available. Yet despite these efforts, questions remain about the Creative Commons project's aims and intentions and the vision of free culture that it offers. These questions become all the more significant as the Creative Commons develops into a more influential and voluble "representative" and public face for libre culture.

We recognise the constructive nature of the work done by the Creative Commons and, in particular, its chief protagonist, Lawrence Lessig. Together they have generated interest in important issues that we hold dear. But here we wish to stand back for a while and subject some of the ideas of the Creative Commons project to interrogation and critique. We don't do this because we think that we have a better understanding of the actions of and motivations of individuals and groups involved in libre culture. In fact, without a great deal of symbolic violence, we think it would be impossible to faithfully represent libre culture in all of its diversity. So rather than attempting to represent what libre culture *is*, an ill-fated and thankless task, we work on the basis of *what it could become*. This isn't a question of mimesis, of Archimedean points, of hermeneutics. It's a question of thinking about libre culture in a more experimental and political way.

Art by Trine Bjørkmann Andreassen

We argue that the Creative Commons project on the whole fails to confront and look beyond the logic and power asymmetries of the present. It tends to conflate how the world is with what it could be, with what we might want it to be. It's too of this time – *it is too timely*. We find an organisation with an ideology and worldview that agrees too readily with that of the global "creative" and media industries. We find an organisation quick to accept the specious claims of neo-

classical economics, with its myopic "incentive" models of creativity and an instrumental view of culture as a resource. Lawrence Lessig is always very keen to disassociate himself and the Creative Commons from the (diabolical) insinuation that he is (God forbid!) anti-market, anti-capitalist, or communist. Where we might benefit from critique and distance, the Creative Commons is too wary to advocate anything that might be negatively construed by the "creative" industry. Where we would benefit from making space available for the political, the Creative Common's ideological stance has the effect of narrowing and obscuring political contestation, imagination and possibility.

A commons without commonalty

Like others before him, Lawrence Lessig bemoans the loss of a realm of freely shared culture. He writes about the colonisation of the public domain brought about by extensions in intellectual property law and the closing down of the technical architecture of the internet. He rightly identifies the way in which global media corporations have lobbied to extend the terms of copyright law so that they can continue to profit from their ownership of creative works. He also identifies the way in which private interests are simultaneously encoding and enrolling digital technologies in order to support their control of artistic and intellectual creativity. Whereas others who problematise these trends turn to the political, the legal professor's penchant is to turn to the field of law and lawyers. What follows is a technical attempt to (re-)introduce a commons by instituting a farrago of new legal licences in the existing system of exploitative copyright restrictions. This is the constructive moment of the so-called "Creative" Commons.

We'll return to this shortly. But first, before getting ahead of ourselves, we should recognise that the action that the Creative Commons project takes is already anticipated in how they represent social reality and define the "problem" in hand. The way in which we construct a problem is also to always render certain beliefs and actions (and not others) obligatory and justified. And so, if anywhere, this is where we must look first.

For us, Lessig's particular understanding of the world, and his desire to strike a balanced bargain between the public and private that follows from this, appear naïve and outmoded in the age of late capitalism. Listen to the political economists. Capital is continually rendering culture and communication private, subject to property rights and the horror of commercial exploitation and beautification. When immaterial labour is hegemonic, the relationship codified in intellectual property between the "public" and "private", between labour and capital, becomes a crucial locus of power and profit. And it is quite natural that private interests would want to protect and extend this profit base at all costs. Their existence depends on it. If libre culture or the Creative Commons threatens this profit base in any way, wars of manoeuvre and position will ensue, where corporations and the state will set out either to crush or co-opt.

The paramount claim of Lessig's prognosis about the fate of culture is that we will be unable to create new culture when the resources of that culture are owned and controlled by a limited number of private corporations and individuals. As far as it goes, this argument has appeal. But it also comes packaged with a miserable, cramped view of culture. Culture is here viewed as a resource or, in Heidegger's terms, "*standing reserve*". Culture is valued only in terms of its worth for building something new. The significance, enchantment and meaning provided by context are all irrelevant to a productivist ontology that sees old culture merely as a resource for the "original" and the "new". Lessig's recent move to the catchphrase "Remix Culture" seems to confirm this outlook. Where culture is only standing reserve it can be owned and controlled without ethical question. The view of culture presented here is entirely consistent with the creative industry's continual transformation of the flow of culture, communication and meaning into decontextualised information and property.

This understanding of culture frames the Creative Common's overall approach to introducing a commons in the information age. As a result, the Creative Commons network provides only a simulacrum of a commons. *It is a commons without commonalty*. Under the name of the commons, we actually have a privatised, individuated and dispersed collection of objects and resources that subsist in a technical-legal space of confusing and differential legal restrictions, ownership rights and permissions. The Creative Commons network might enable sharing of culture goods and resources amongst possessive individuals and groups. But these goods are neither really shared in common, nor owned in common, nor accountable to the common itself. It is left to the whims of private individuals and groups to per-

mit reuse. They pick and choose to draw on the commons and the freedoms and agency it confers when and where they like.

We might say, following Gilles Deleuze, that the Creative Commons licensing model acts as a "*plan(e) of organisation*". It places a grid over culture, communication and creativity, dividing it and cutting it into discrete pieces, each of which have their own distinct licence, rights and permissions defined by the copyright holder who "owns' the work. Lessig's attempt to make it easier to understand which creative works can, or cannot, be used for modification (due to copyright) has spawned a monster with a thousand heads. The complexity of licences and combinations of licences in works has expanded exponentially.

This plane of organisation ensures that legal licences and lawyers remain key nodal and obligatory passage points within the Creative Commons network, and thereby constitute *blockages* in the flow of creativity. But what is happening is that the ethical practice of sharing communication and culture is being conflated with a legal regime that seeks bureaucratically to enforce the same result through comprehensively drafted and dense legalese. At least Richard Stallman and his ingenious GNU General Public License (GPL) is honest in claiming to be an ethical rather than purely legal force. The GNU GPL has tenacity not due to its legal form alone. The GPL is based on a network of ethical practices that continually (re-)produce its meaning and form. The commons is always more than a formal legal construct. *The commons is based on commonalty*.

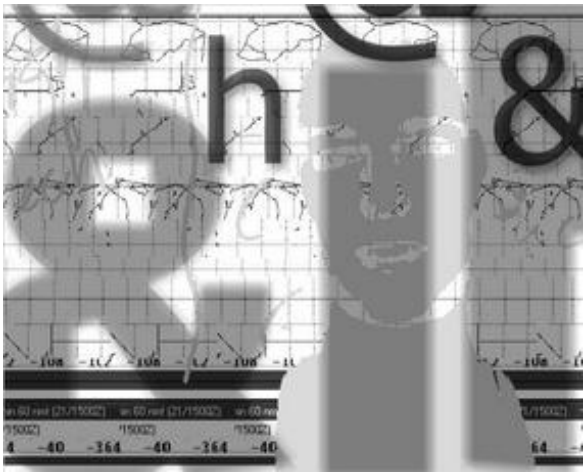
Very simply put, the commons has historically been understood as something shared in common. In pre-capitalist times the commons were referred to as "*Res Communes*". This included natural things that were used by all, such as air and water. This ancient concept of the commons can be traced through Roman law into the various European legal systems. Through migration and colonisation, it can also be found in the United States and other countries around the world. In the UK, there's still the concept of common lands, albeit a pale shadow of what went before. In the United States, the concept of public trust doctrine is an application of the ancient idea of the commons. To a certain extent the commons, as *Res Communes*, lies outside the property system. It is separate from both private (*Res Privatae*) and state (*Res Publicae*) ownership. Through copyright the Creative Commons attempts to construct a commons within the realm of private ownership (*Res Privatae*). The result is not, dare we say it, a commons at all. The commons are formed through commonalty and common rights, resistant to any mechanisms of privatisation, whether those of the Creative Commons or not. Without commonalty, without the common substrate through which singularities act, live and relate, there could be no commons at all.

A commons with commonalty

The marketing and PR of the creative industries, their lobbying attempts and their lawyers, have not managed to persuade us that they are true friends of creativity. They don't convince us of their specious incentive claims nor of the idea that sharing knowledge, concepts and ideas is criminal. If anything, property is the corruption and the crime: an act of theft from the common substrate of creativity. But still global media corporations continue to work to transform the system — legally, technologically and culturally — to facilitate their ownership and control of creativity. This is a social-factory of immaterial labour where all of life — loving, thinking, feeling and sharing — is subject to the corruption of privatisation and property.

As we've already suggested, the commons is an ethical and not just a legal matter. We underscore the point. The commons rests on commonalty, on ethical practices that emerge rhizomatically through the actions, experiences and relations of decentralised individuals and groups, such as the free/libre and open-source movement. For this reason, libre culture is far more than just a protest movement. It is not only reactive; it is productive. It creates new forms of life through its practices. It creates new possibilities. Yet, in our view, there has to be a political dimension to libre culture as well. This expresses itself through political imagining, action and a broader struggle for true democracy. And, as such, it is important to recognise the damage that could be done to libre culture by those spokespeople who seek to depoliticise it. In the world in which we find ourselves, political awareness, resistance and struggle are essential in order to defend the idea and practice of a creative field of concepts and ideas that are free from ownership — to stand up, that is, for the commons and commonalty. It is to the political struggle of libre culture and the commons that we finally turn.

Art by Trine Bjørkmann Andreassen



Where is the politics of libre culture to be found? The answer: at numerous levels. Political struggle will no doubt be orientated towards the nation state (as Maureen O'Sullivan argued in "*A law for free software*" in issue 2 of Free Software Magazine). For the time being at least, nation states are obligatory passage points that retain a privileged position in upholding and enforcing law. But it cannot remain there alone. The commonalty of creativity shows little regard for national boundaries and, of course, neither does the global reach of the profiteers from the creativity and media industry. Creativity is at once too small and too large. Political action and the struggle for true democracy will have to also be aimed simultaneously at local and global levels. For the latter, we might envisage a treaty obligation through measures such as preventing the commodification of human DNA and life itself or a UN protectorate to defend the sanctity of ideas and concepts. We might picture something akin to Bruno Latour's "*Parliament of Things*", a space where not just the human is represented, but all of life has a defender, all of life has a voice.

Law is a juridico-legal grid placed on social life. This grid is upheld and enforced by a network of states and other forces of governance and governmentality. Reliance on law and the state makes the legal licences of the Creative Commons (or other legal versions of the commons for that matter) vulnerable and precarious. We cannot be sure, as yet, how Creative Commons licences will stand up in legal practice. For they have not been properly tested. But there is one thing of which we can be relatively sure. In principle, we might all be equal in the eyes of law. In principle, the ladder of the law might not have a top or a bottom. But, in practice, economic power matters. We know that law and the state are not immune to economic persuasion, to lobbying, to favours and so forth. And, because of this, the commons remains subject to the threat and corruption of privatisation and commodification.

We do not want to suggest by this that all legal and public rights, including the protection of the commons by the state or global institutions such as the UN, are worthless. This would be a perversion of our position. What we would stress is that such rights originate with the people through political struggle, not with legislators or legal professors setting them down on pieces of paper. And if these rights are to be maintained, if a commons is to be instantiated and protected, there is a need for political awareness, for political action, for democracy. Which is to say, any attempt to impair commonalty and common rights for concepts and ideas must meet resistance. We need political awareness and struggle, not lawyers exercising their legal vernacular and skills on complicated licences, court cases and precedents. We're sorry to say, however, that this does not appear to be a political imaginary (and political struggle) that the Creative Commons project shares or supports.

Copyright information

© 2005 by David M. Berry, Giles Moss

This article is made available under the "Attribution-Share-alike" Creative Commons License 2.0 available from http://creativecommons.org/licenses/by-sa/2.0/.

About the author

David Berry is a researcher at the University of Sussex, UK and a member of the research collective The Libre Society (http://www.libresociety.org). He writes on issues surrounding intellectual property, immaterial labour, politics, free software and copyleft.

Giles Moss is a doctoral student of New College, University of Oxford. His research interests span the field of social theory, but he currently works on the intersections of technology, discourse, democratic practice and the concept of the "political".

ALL MUSIC AND ARTWORK IS
RELEASED UNDER CREATIVE COMMONS
ATTRIBUTION SHARE-ALIKE LICENSE

WWW.LIBREESOCIETY.ORG
WWW.CREATIVECOMMONS.ORG
WWW.WIKIPEDIA.ORG

E-LOCA-TRONICA 2005
STICK TO WHAT YOU DON'T KNOW

HTTP://WWW.LOCARECORDS.COM

INFO@LOCARECORDS.COM

YOU CAN DOWNLOAD MUSIC,
VIDEOS AND TEXT BY VISITING
WWW.LOCARECORDS.COM

LOCA RECORDS, PO BOX 233,
BRIGHTON BN2 3WX, UK

Starting an e-business?

- Do it right
- Do it offshore
- Do it with us

Save yourself from a mountain of paperwork. Open your company in the British Virgin Islands with us, the British Virgin Islands experts. Manage your company's bank account. Manage your life.

Di Bernardo Massimo
OFFSHORE

90 Main St - Road Town, Tortola
British Virgin Islands

http://www.dbm-offshore.com
info@dbm-offshore.com